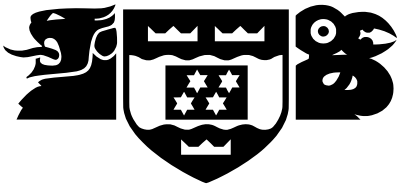
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Multifaceted Web Service Composition

Alexandre Sawczuk da Silva

Supervisors: Dr. Hui Ma, Prof. Mengjie Zhang

May 20, 2015

Submitted in partial fulfilment of the requirements for PhD.

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 respect. 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 are: (1) fully functional (i.e. fully executable, with semantically-matched inputs and outputs), (2) respect composition constraints (e.g. user can specify logical branching), (3) are optimised according to non-functional Quality of Service (QoS) measurements, and (4) respond to changes in a dynamic environment. The overall goal of this thesis is to propose a hybrid Web service composition approach that considers elements from all four facets described above when generating solutions. This approach combines elements of AI planning and of Evolutionary Computation to allow for the creation of compositions that meet all of these requirements.

Contents

Figures

Chapter 1

Introduction

1.1 Problem Statement

As applications increasingly interact with the Web, the concept of Service-Oriented Architecture (SOA) [64] emerges as a popular solution. The key components of SOA are Web services, which are functionality modules that provide operations accessible over the network via a standard communication protocol [30]. One of the greatest strengths of Web services is their modularity, because it allows the reuse of independent services that provide a desired operation as opposed to having to re-implement that functionality. The combination of multiple modular Web services to achieve a single, more complex task is known as *Web service composition*. 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 may be connected to each other, with the outputs of a service satisfying the inputs of the next, starting from the composition's given overall input and finally leading to the composition's required output.

An example of an application of Web service composition is ENVISION (Environmental Services Infrastructure with Ontologies) [55], 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 [55]. ENVISION is meant to be used by scientists who are experienced with geographic models but do not have a technical computing background, therefore the motivation of this work 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 [50]. 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 highlighted that not a single discovery tool from the literature was 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. Another problem is that the services used in a composition may produce too much data, or incur too much latency, meaning that the final composition is slowed down as a result. Standardising and automating the service composition process would eliminate the need for dealing with these difficulties manually, thus developing a system capable of creating compositions in a fully automated manner is one of the holy grails of the field [57].

Automated Web service composition is complex, and in fact it is considered an *NP-hard problem*, meaning that the solution to large tasks is not likely to be found with reasonable computation times [60]. Due to this complexity, a variety of strategies have been investigated in the literature, focusing 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 this abstract process is not typically required and instead the focus is on discovering the connections between services that lead from the task's provided output to its desired [60]. In workflow-based approaches, the abstract functionality steps necessary to accomplish the task requested by the user have 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 translated into an optimisation problem where the objective is to achieve the best overall composition quality. This optimisation is often performed using *Evolutionary Computation* techniques. However, the disadvantage is that a 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 quality-based optimisation on the selected services when using planning.

The overall goal of this thesis is to propose a Web service composition approach that hybridises elements of AI planning-based approaches and Evolutionary Computation workflow-based approaches, enabling the construction of a workflow according to user constraints as well as the optimisation of that workflow according to the quality of its composing services. This new approach tackles several aspects of Web service composition, such as the use of multi-objective optimisation, semantic Web service selection, and dynamic Web service composition.

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**, services must be combined so that their operation inputs and outputs are properly linked, i.e. the output produced by a given service is usable as input by the next services in the composition, eventually leading up to the desired overall output. As the **second facet**, services must be arranged appropriately in the composition according to the desired outcome (e.g. in sequence, in parallel). Particular attention must be paid to conditional constraints, when the composition is required to have multiple execution options – branches – according to a given condition [79, 74, 38]. 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 service modules, and the reliability of those modules. As the **fourth facet**, the environment of the composition must be recognised as dynamic, with QoS values that fluctuate and services that become unavailable/available over time. These facets are discussed in more detail below:

1. **Functionality:** Functional solutions are compositions where connected 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. In order to ensure functionality, two main approaches can be used: *immediate* and *gradual*. Whenever a solution is built using immediate approaches, this solution is fully functional, i.e. the inputs of all atomic services are fully satisfied, and so are the overall task outputs given the provided inputs. This is typically done by employing AI planning algorithms that verify the correctness and completeness of the connections between services at every building step [79]. When a solution is built using gradual approaches, on the other hand, there are no guarantees that it will be fully functional. As opposed to performing checks at every building step, gradual approaches rely on the notion of improving solutions over multiple iterations, each time penalising incorrect and/or incomplete connections between services. In practice, these approaches are typically implemented using Evolutionary Computation techniques with penalties enforced through the fitness function [70]. The simplest way of establishing that the inputs of an atomic service have been satisfied is by verifying that there is an exact type match between each target input and its corresponding incoming connection (i.e. the output of a previous service). However, a more sophisticated matching approach relies on measuring the semantic similarity between the output and input types in question with the help of an ontology [9]; in this case, the smaller the distance between the two values, the better the match.

2. **Composition constraints:** In addition to considering the functionality of services within the composition, it is necessary to arrange them appropriately. This ensures, for example, that services that depend on each other are sequentially connected, while services that are independent from each other are allowed to be executed in parallel. Another fundamental construct is a conditional constraint, which determines when the execution of a composition should branch into one of multiple possible paths, depending on runtime values. For example, consider the scenario of an online book shopping system, adapted from [79]. In this scenario, the objective is to employ existing Web services to accomplish a basic book shopping operation. Preferably, the services to be used, the order in which they are to be invoked, and how they interact with each other should be determined automatically. Therefore, the book and customer details (e.g. title, author, customer information) and the expected purchase outcome (e.g. receipt) act as the composition task inputs and outputs, and the shopping-related services as the atomic composition components. In certain cases, however, the customer may have specific constraints. For example, the customer's preferred method of payment is likely to depend on their current account balance: if the customer has enough money to pay for the book in full, then they would like to do so, otherwise the customer would like to pay by installments. In this case, the composition task has one set of inputs (book and customer details), and a condition (balance) that may lead to two different sets of outputs depending on whether it is met (either a receipt for paying in full or the initial installment bill). In this type of composition, the runtime value of the type in the condition is used to ultimately decide which set of outputs should be produced. Different techniques have been explored to achieve compositions that

consider conditional constraints, including the use of AI planning with rules encoding user constraints [9] and the representation of the composition as a constraint satisfaction problem to be processed with a solver engine [38]. This territory has not been widely explored by applying Evolutionary Computation (EC) techniques; yet, due to their flexibility and efficiency, it would be interesting to focus on the investigation of ways in which to extend them to apply these constraints.

3. **Quality of Service:** Quality of Service (QoS) refers to the non-functional (quality) attributes associated with a Web service, such as its expected execution time when answering requests, its availability, and its scalability [42]. 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 composition's QoS by selecting the right combination of constituting services. As mentioned earlier, the selection of a set of services in order to maximise the overall QoS can be mapped into a classic optimisation scenario, and this has been done extensively in the literature. A variety of EC techniques have been used in this context [78], as well as other optimisation techniques such as integer linear programming [88]. Interestingly, AI planning techniques were also used to this end, even though in this case they are incapable of also considering conditional constraints [21]. In the realm of QoS optimisation, multi-objective techniques are often employed to the problem of Web service composition, since the optimisation of potentially conflicting QoS attributes such as time and cost is more intuitively performed using independent objective functions [49]. An extension to the problem of QoS-aware composition is that of SLA-aware Web service composition [87], where solutions must not only be quality-optimised but also abide by minimum quality standards (i.e. thresholds) defined by the composition requestor. SLA-aware composition has focused on tasks with a single execution path, but research still must be conducted on those with multiple execution paths (i.e. tasks with conditional constraints).

4. **Dynamic composition:** In a more realistic scenario, Web service compositions exist within a dynamic environment where the quality and availability of the atomic services in the repository varies as time passes. In such a dynamic environment, providing a static composition solution to a task is no longer enough, since this solution may decrease in quality throughout time and/or become non-executable if some of its composing services go offline. Thus, the focus of dynamic Web service composition is on monitoring and updating composition solutions as they become outdated [44]. The majority of techniques aimed at updating outdated or faulty compositions rely on building solutions that present constructs allowing for dynamic adaptation [2], but not many EC-based approaches have been tried in this area.

The above discussion refers to several techniques that have been proposed to address the composition problem [69]. These techniques produce promising results, however they do not account foll all of these composition facets at once. For example, AI planning techniques for composition [33, 21] focus on guaranteeing functional correctness (first facet), and either fulfilling conditional constraints (second facet) or optimising QoS (third facet); similarly, EC techniques such as Genetic Algorithms (GA) and Genetic Programming (GP) [70, 78] focus on QoS in addition to functional correctness, but do not include the fulfilment of conditional constraints.

1.2.1 Limitations of Current Web Service Composition Approaches

Hybridisation of AI planning and EC composition Techniques

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 outstanding at ensuring the creation of composition solutions that have appropriate connections between the outputs and inputs of the composing services, and also at ensuring the creation of branches according to the constraints provided. On the other hand, EC techniques are ideal for encountering solutions with a good overall quality amongst a very large array of possibilities. 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 capabilities, and thus considers more service composition facets simultaneously. In this hybrid approach, users would be able to specify the conditions of when branching should occur, the order in which these conditions should be observed, and the outputs each execution branch should produce; the technique would then return a suitable solution. Despite being a promising approach, this area has currently not been investigated in depth.

Multi-Objective Composition Optimisation

Existing optimisation approaches have focused on optimising the QoS of composite services that have only one execution path (i.e. there are no conditional constraints leading to different execution outcomes), by relying either on a single or on multiple objective functions. In the case of single-objective approaches, the different QoS attributes are combined through a weighted sum that produces a unified quality score used for ranking candidate solutions; multi-objective approaches, on the other hand, evaluate each QoS attribute using a separate score and divide candidate solutions into groups by comparing each of these dimensions simultaneously. In both types of optimisation, SLA constraints have been employed to restrict the quality of solutions to acceptable levels. When optimising compositions with multiple execution paths (i.e. with conditional constraints), a solution's overall quality is typically calculated by aggregating the quality values of each execution path according to a weighted sum, where weights reflect the probability of each path being executed. This approach has the effect of attenuating outlying quality values, which may be problematic for SLA-aware composition because it masks particular execution paths where quality thresholds have not been met. To overcome this problem, solutions with multiple execution paths should have the quality of each path optimised and verified independently, an approach that has not yet been explored by current researchers.

Semantic Web Service Selection

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 inexact matches have been investigated. In a typical 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 concept of an abstract service cannot be used when selecting services during an AI planning-based composition, which decreases the efficiency of this approach. As this problem has not been addressed in current works, a better way of performing semantic selection must be proposed for use with AI planning-based composition approaches.

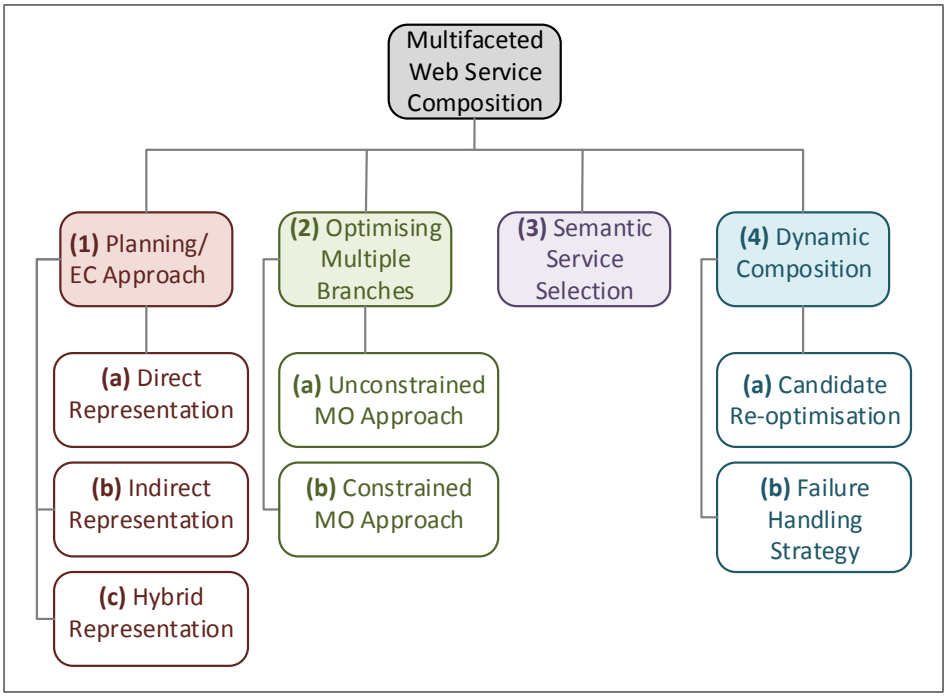


Figure 1.1: Research objectives and sub-objectives.

EC-based Dynamic Web Service Composition

Evolutionary Computation (EC) approaches to Web service composition have been extensively investigated in static scenarios, where the quality and availability of services are assumed to remain constant, however the use of these techniques has only been superficially explored in dynamic scenarios. Despite this lack of research, EC-based approaches show promise in the area of dynamic Web service composition for two reasons: firstly, they allow a composite service system to *self-heal* by maintaining a population of solutions and using them as alternative compositions in the case of failure; secondly, they support *dynamic adaptation* by allowing solutions to be further evolved in order to account for changes in the QoS values of atomic services. Given these advantages, it is desirable to investigate the use of EC approaches in a dynamic composition context.

1.3 Research Goals

The overall goal of this thesis is to propose a hybrid Web service composition approach that considers elements from all four facets described above when generating solutions. More specifically, this approach combines elements of AI planning, to ensure functional correctness and constraint 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 that require multiple execution paths. The research goal described above can be achieved by completing the following set of objectives, outlined in Figure 1.1, which are intended to be used as research guides throughout this project:

1. Create a hybrid QoS-aware Web service composition technique that combines elements of AI planning and evolutionary computation to allow for the optimisation of composition solutions with conditional constraints. One of the key aspects of this tech-

nique is the representation of each composition solution. Undoubtedly the simplest model would be that of a linear vector, with each element being a service that should be included into the composition, however this linear structure does not satisfactorily encode the relationships and links between different services. Another problem with a vector is that the EC techniques which use such a representation do not allow for structures of varying lengths, a requirement when performing fully automated composition. The search for a suitable solution representation leads to the following three sub-objectives:

(a) *Propose a direct solution representation for the planning and EC-based Web service composition technique.*
A tree or graph representation is well-suited to represent Web service composition solutions with conditional constraints, since such structures are naturally capable of representing the links between the composing services and multiple execution paths. These structures may be referred to as *direct representations* of solutions, since the genotype and the phenotype of each solution are the same, meaning that solutions are represented directly as they should be interpreted. Proposing a representation based on these structures involves some trade-offs that must be carefully considered, such as the existence of EC techniques that support that structure and the computational cost they incur.

(b) *Propose an indirect solution representation for the planning and EC-based Web service composition technique.*
Despite the intuitiveness of direct representations, it has been argued in the problem domain of scheduling that indirect representations, where the final result must be decoded from a population candidate, have been shown to have better search performance than their direct counterparts [32, 16]. Due to this evidence, an indirect candidate representation should be proposed and compared to the direct representation. This indirect representation could be based on the Web service composition Particle Swarm Optimisation (PSO) approach proposed in [18].

(c) *Propose a hybrid solution representation for the planning and EC-based Web service composition technique.*
In case there is a trade-off between the execution time and the quality of the solutions produced by the direct and indirect representations developed in Objectives 1a and 1b, a hybrid representation should be proposed to combine the strengths of these two approaches. This representation could comprise the structure used in the direct representation with the addition of weights from the indirect representation. Comparisons should be performed to determine whether the hybrid solution presents any performance or quality gains.

2. Develop a many-objective (MO) approach to optimising the quality of candidate compositions with multiple execution paths, abiding by SLA constraints. Two factors must be taken into account when optimising Web service compositions with multiple execution paths: firstly, each Quality of Service (QoS) attribute must be optimised independently, since they may be conflicting with each other; secondly, each path of the composition must also be independently considered, since the QoS values for each path may vary significantly. The development of this approach can be divided into the following two sub-objectives:

(a) *Propose an unconstrained MO approach for independently optimising the execution paths of a Web service composition solution with conditional constraints.*

The challenge in optimising compositions with multiple execution paths, while at the same time considering several independent quality measures, is the number of dimensions that must be considered simultaneously. On the one hand, encoding each individual quality measure of each execution path as an independent value provides the a very expressive representation of the problem; on the other hand, MO optimisation with a large number of dimensions may result in a solution set that contains many unremarkable solutions. Thus, the proposed approach must handle this issue.

(b) *Extend this MO approach to also consider SLA constraints.*
Once the fundamental MO optimisation approach has been proposed, it should be extended to observe SLA constraints. Note that these constraints must be enforced for each execution path individually, to ensure that all runtime options have been optimised according to the quality parameters of the composition requestor.

3. Develop an EC technique for performing semantic Web service selection in the context of a planning-based composition technique. As discussed in Objective 1, the composition technique investigated in this thesis combines elements of AI planning and of EC approaches to achieve compositions that are correct, have conditional constraints, and can be optimised according to their QoS attributes. In planning-based approaches to service composition, however, the semantic selection process of candidate atomic services is inefficient, since there are many possible service matches to consider at each planning step. In fact, it may be unfeasible to consider all possible service matches in an exhaustive manner when handling larger service repositories, which invites the use of non-exhaustive approaches such as EC techniques to accomplish this task. Thus, this objective entails the use of an optimisation technique to discover semantically matching services to be included in a composition candidate. Note that this objective aims to investigate the use of an optimisation technique nested within the service selection step of the planning/EC approach.

4. Modify the planning/EC composition technique to work in a dynamic environment. In a static scenario the planning-EC technique is executed once for a given composition task, returning a composite result under the assumption that the quality levels and the availability of the atomic services included in that result will remain constant. In a more realistic dynamic scenario, however, the quality of the services in the repository may fluctuate and occasionally services may become unavailable. To account for these setbacks, solutions must be corrected and updated in response to changes in the environment. In order to do this, the planning-EC technique is to be modified to retain a population of candidates as alternatives in case of failure, and further generations are to be evolved as the QoS values of services in the repository change, thus leveraging the natural features of Evolutionary Computation. These improvements are to be performed in two steps:

(a) *Extend planning/EC technique to re-optimise candidates as the quality values of services changes.*
Usually, EC approaches dedicated to Web service composition create a population of candidates, optimise this population, and identify the best solution candidate, discarding the others in the process. In a dynamic scenario, disposing of the remaining candidates would be wasteful, since some of these alternative solutions may become promising as quality values change. Instead of destroying the population, the extended technique should maintain it for future re-optimisation. A

key challenge in this approach is striking a balance between population diversity and effective optimisation.

(b) *Create a strategy for handling service failure using other candidates in the population.* In addition to quality changes, the atomic services used in a composition may occasionally fail/become unavailable. An alternative execution plan must be used to prevent the composite service from being impacted, and the proposed solution is to select as efficiently as possible an unaffected candidate from the population as the replacement.

1.4 Published Papers

During the initial stage of this research, some investigation was carried out on the suitability of different candidate representations for EC-based Web service composition. This culminated in the publication of the following papers:

- Alexandre Sawczuk Da Silva, Hui Ma and Mengjie Zhang. "A GP Approach to QoS-Aware Web Service Composition and Selection". *Proceedings of the 10th International Conference on Simulated Evolution and Learning (SEAL 2014). Lecture Notes in Computer Science*. **Vol. 8886**. Dunedin, New Zealand. December 15-18, 2014. pp. 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 2015 IEEE Congress on Evolutionary Computation (CEC 2015)*. Sendai, Japan. 25-28 May, 2015 (To Appear)
- 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 (Accepted)

1.5 Organisation of Proposal

The remainder of the proposal is organised as follows: Chapter 2 provides a fundamental definition of the Web service composition problem and performs a literature review covering a range of works in this field; Chapter 3 discusses the preliminary work carried out to explore the hybridisation of AI planning techniques and EC-based techniques for Web service composition, one of the key ideas proposed in this project; Chapter 4 presents a plan detailing this project's intended contributions, a project timeline, and a thesis outline.

Chapter 2

Literature Review

This chapter begins with an overview of the field of Web service composition in Section 2.1, then it addresses several areas of current research interest. Section 2.2 discusses single-objective compositions, both biologically and non-biologically inspired approaches; Section 2.3 delves into approaches that optimise several dimensions independently (i.e. multi-objective, many-objective, top-K); Section 2.4 discusses AI planning-based approaches to service composition; Section 2.5 covers semantic approaches to Web service selection; Section 2.6 discusses the issue of dynamic Web service composition. Finally, Section 2.7 presents a summary of the important points identified in this review, alongside a discussion of the limitations of existing approaches.

2.1 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, given that there are multiple services offering the same functionality. They 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 tasks. A workflow-based Web service composition approach can usually be decomposed into a series of steps, reflecting the process required to produce a solution [60]. These steps are shown in Figure 2.1 and discussed below:

1. *Goal specification:* The initial step in a Web service composition 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. This workflow is generated by the user and is referred to as *abstract*, since it contains a series of tasks that can be accomplished by employing a number of different existing Web service implementations. In addition to this workflow, the QoS requirements are determined based on the user's preferences.

2. *Service discovery:* Once an abstract workflow and a set of preferences have been provided, the next step is to discover candidate concrete services that are functionally and non-functionally suitable to fill the workflow slots in a service repository. The focus at this stage is to find candidates that provide the functionality required to fulfil the tasks, regardless of their quality levels.

3. *Service selection:* After pools of candidate services have been identified for each workflow slot, a technique is employed to determine which discovered services best fulfil each slot. The result of this process is the creation of a concrete Web service composition.

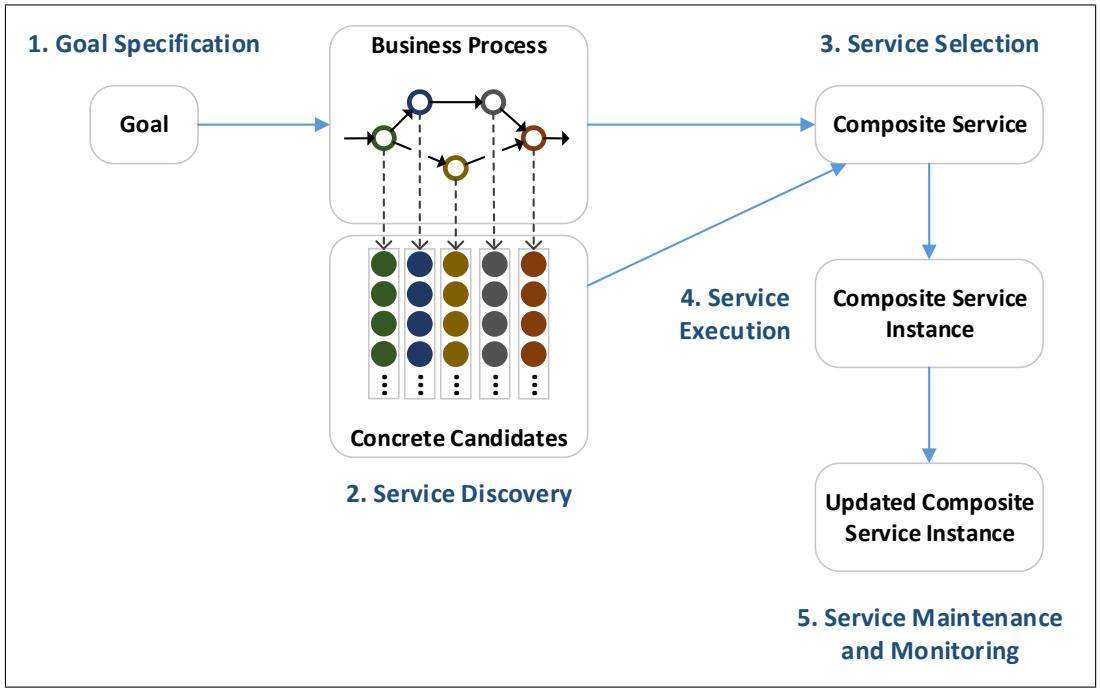


Figure 2.1: Typical steps in a workflow-based automated Web service composition solution [60].

4. *Service execution:* The creation of the composition is followed by the execution of an instance of this composed Web service.

5. *Service maintenance and monitoring:* During execution, the created instance is constantly monitored for failures and/or changes to the composing atomic services, and corrective actions are dynamically carried out as necessary.

It is important to draw a distinction between semi-automated and fully automated composition approaches [69]. The steps discussed above are typical of a **semi-automated approach**, where an astract workflow is provided in the goal specification stage and the composition algorithm is only required to complete the abstract slots of this workflow. In a **fully automated approach**, on the other hand, an abstract workflow is not provided during the goal specification stage, and instead is calculated at the same time that services are selected based on user preferences such as the desired overall composition inputs and outputs. 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 [18], thus they are the focus of this project. It must also be noted that this work is focused on exploring new techniques to performing service selection, but not service discovery, maintenance, and monitoring.

Besides identifying the atomic Web services most suitable to fulfilling key composition tasks, the selection process often takes additional user constraints into account. A survey of the literature in the area shows that two types of contraints are commonly taken into account. The first group comprises the creation of service compositions that are optimised according to the Quality of Service (QoS) constraints on its constituent atomic services, where QoS attributes may be thought of as features that indicate the quality of a given Web service, such as the time it requires to return a response and its financial cost of execution [18]. The majority of works in this area use maximising and minimising functions for the dif-

ferent QoS attributes, meaning that they attempt to obtain solutions with the best possible qualities without any thresholds [81, 62, 88, 27, 92, 78, 60, 39, 45]. However, there also exist approaches focused on what may be referred to as a Service Level Agreement (SLA)-aware optimisation, which is solutions must meet certain predetermined QoS thresholds in order to be considered valid (e.g. the financial cost of each service used in a composition must not exceed 50 dollars) [88, 87, 13, 6].

The second group comprises the creation of service compositions that have multiple execution branches, indicating user preferences at runtime [79, 10, 38, 53, 7]. For example, the output of a service *A* determines which service to execute next; if the output is greater than a certain threshold, then service *B* should be executed; otherwise, service *C* should be executed instead. These preferences are expressed logically as conditions, and affect the workflow used to establish the connections between services rather than the individual services themselves. These two types of user constraints are discussed in more detail in subsequent Sections in this chapter.

2.1.1 Web Service Composition in Practice

The research material published on Web service composition is highly theoretical and frequently employs layers of abstractions and simplifications intended to make the problem at hand manageable. However, it is also important to investigate how these ideas fit into practice, which is the objective of this Subsection. As mentioned earlier, while significant amounts of work are being performed in the field of automated Web service composition, these approaches ultimately remain on the theoretical level due to difficulties that have not yet been fully addressed. On the one hand, a study [50] has shown that the composition of services is fraught with issues. A central problem is that there are discrepancies between the concepts used by different services: the schemas used differ from each other, even if the services handle exactly the same domain. Another problem is the existence of services that produce too much data, low-quality data, or that incur too much latency. These characteristics may slow down the execution of the composition and even require human intervention, defeating automation efforts.

On the other hand, an automated framework for Web service compositions which can be applied to real services has already been proposed [52]. The functionality of this framework was demonstrated by creating a composition that uses the Amazon Virtual Cart service, the Amazon Book Search service and a bank's Point of Sale Web service. The authors of the framework point out that it is very challenging to find service documentation that is comprehensive enough, and that functionality details had to be identified from natural language explanations intended for developers who are working manually. Additionally, the specification of control flow requirements must be performed manually using a logic programming language. Despite these difficulties, the evaluation of this framework shows that a composition can be found within seconds when using it as opposed to an estimated 20 hours of manual development, showing that research on Web service composition provides some immediate benefits.

These two works provide opposing views on the viability of automated Web service composition, however a more recent approach [59] presents the interesting middle ground of user-centered design. This systems combines manual and automated techniques, providing a browsing tool that allows users to explore the repository and gain some understanding of the offered QoS value ranges of services before having to write any QoS requirements, as users may often be unaware of what a reasonable QoS value would be for a given dataset. Users are also supported through the selection process by utilising a UI that diminishes their cognitive overload, and helps them express their requirements using a standardised

language. In addition to these tools, this approach also proposes a clustering algorithm that groups service candidates according to their range of QoS values, with the objective of providing selection options when users have fuzzy (i.e. soft) requirements. The strength of this approach is that it does not undervalue human intervention in the composition process, instead providing tools that empower system users. As Web service composition always requires some degree of user involvement, at the very least when setting composition requirements, this type of approach may prove to become an increasingly popular composition solution.

2.2 Single-Objective QoS-Aware Composition Approaches

This section discusses composition approaches that optimise solutions according to their overall QoS. These approaches can be divided into biologically inspired methods, which use Evolutionary Computation to reach their goal, ang general optimisation approaches that do not turn to evolutionary techniques. These two groups are quite distinct, but they have the commonality of using an objective function as the guide with which to measure the quality of the candidate solutions.

2.2.1 Biologically-Inspired Approaches

Biologically-inspired Web service composition approaches rely on evolutionary computation algorithms, which implement their search strategies by drawing inspiration from nature, namely the behaviour of social animals such as bees, fish, and ants. An important distinction between these different bio-inspired approaches is in their representations of the composition problem. These varying representations are discussed throughout this Subsection, and visually summarised in figure 2.2. **Genetic Algorithms (GA)** are a popular choice for tackling combinatorial optimisation problems, and thus have been widely applied to the problem of Web service composition [78]. The encoding scheme for a composition is commonly done as a vector of integers, where each integer corresponds to a candidate Web service, even though some authors have attempted to use matrix representations that also include semantic information about services. A population of candidates is evolved for several generations using generic 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 crossover is modified at random in order to introduce some genetic diversity. 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)** bears similarities with GA, also relying a on vector representation for candidates. However, 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. As PSO may also present the problem of not fully optimising solutions (i.e. converging prematurely), hybrid approaches have been attempted to improve its efficiency and optimisation power [78]. A key limitation of both GA and PSO is in the underlying vector representation used by candidates, since it makes it very challenging to encode workflow information and thus perform any type of fully automated Web service composition.

Ant Colony Optimisation (ACO) has also been proposed as a solution to QoS-aware Web service composition [92]. ACO is particularly suitable to a directed acyclic graph (DAG) representation, in other words, the workflow composition representation commonly used in the field. In ACO, the Web service workflow 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

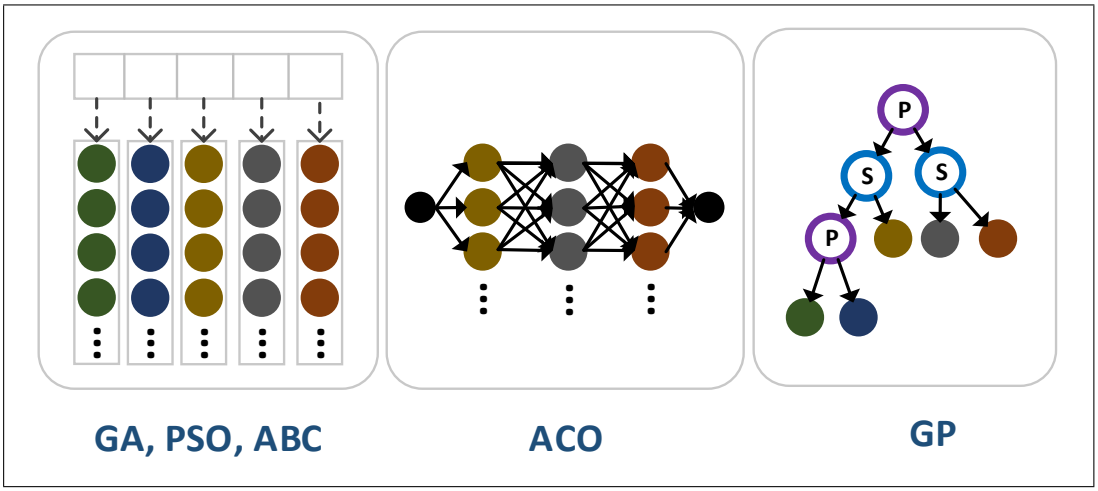


Figure 2.2: The different problem representations employed by biologically-inspired Web service composition approaches.

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 edges of those paths. Meanwhile, the pheromone level of all edges gradually decreases (i.e. evaporates) after each tour of the ants. The graph 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 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. For each concrete Web service, the heuristic factor is calculated based on its QoS values, and the fitness function also measures QoS attributes. As with GA and PSO, this representation also amounts to the idea of semi-automated Web service composition, even though in this case the encoding of workflow information leading to a fully-automated approach would seem to be trivial.

The works in [92] and [81] apply the **Artificial Bee Colony (ABC)** algorithm to Web service composition. The ABC algorithm simulates the behaviour of bees search for food sources. The position of food 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 does not present any food sources that are better than the original. The roles of the bees change according to the colony's needs, which is a feature unique to this algorithm. One of the issues with this approach, as pointed out by the authors, is the search space it explores. The problem with the typical search space for Web service composition is that it is organised based on the proximity of the components included into the composition. For example, given two adjacent solutions a and b in the search space, a will only have one service that is different from all the services included in b. Despite being neighbours, however, the fitness values of a and b may be radically different, and as the optimisation occurs according to a fitness function, this means that the search space is not entirely continuous. These works modify the traditional ABC algorithm to take this problem into account, filtering the neighbourhood of each solution during the search and excluding radically different neighbours from consideration.

Genetic Programming (GP) is, from this project's perspective, possibly the most inter-

esting technique for the problem of Web service composition. That is because GP, unlike the previously discussed techniques, is capable of supporting fully automated Web service composition. In GP approaches [3, 70], workflow constructs are typically represented as the GP tree's non-terminal tree nodes while atomic Web service are represented as the terminal nodes. In this context, workflow constructs represent the output-input connections between two services. For example, if two services are sequentially connected, so that output of service A is used as the input of service B, this would be represented by a sequence workflow construct having A as the left child and B as the right one. 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 configurations allowed in the tree to feasible solutions only. A fitness function is employed for the QoS optimisation of candidates, and 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. One of the difficulties of tackling the problem of Web service composition using GP is that it does not intrinsically support the use of constraints [16], meaning that even if all candidates in a population meet the feasibility constraints, there is no guarantee that subsequent generations will maintain conformance to them. 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, 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. Indeed, the tree representation of an underlying workflow composition may pose difficulties whenever constraint verification is necessary.

Graph Variations of GP

Genetic programming using candidates constructed as graphs (instead of trees) would be ideal for the problem of Web service composition, since dependencies between services could be intuitively encoded. Even though variations of GP with graph candidates do exist, they have not been employed in this domain, therefore the focus of this section is on the techniques and not on the composition problem. Galvan-Lopez [25] proposes a general-purpose EC technique that is modification of the usual GP tree reprentation, allowing the creation of graphs. The key extension proposed here is the addition of pointer functions to certain non-terminal nodes, meaning that they can have connections to nodes in independent subtrees. Program inputs are provided as terminal nodes (similarly to the output-as-root representation discussed earlier), but instead of having a single output location at the root node, they may have other output locations as well, inserted as necessary amongst the non-terminal nodes of the tree. Since the overall tree structure is maintained, it becomes possible to perform the crossover and mutation operations similarly to their implementation in the case of a simple tree. The main difference is that any pointers present in the original tree may not be modified when these operations are applied. In order to ensure that the number of outputs in a tree remains correct throughout these genetic operations, each node in a tree is classified beforehand to establish which ones may be selected for the crossover operation. This representation makes it easier to evolve graphs, however it does not support strongly-typed GP, which is of interest to our research. In addition, each tree is still required to have a single root, meaning that there are connections between the different output nodes. In our problem domain, this is a hindrance because the output nodes must be completely independent from each other.

Miller and Thomson [58] present Cartesian Genetic Programming (CGP), a popular technique for evolving graph structures. The simplicity of SGP 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 obey 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 onward, 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, thus restricting the range of problems it can be applied to. Another problem 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.

Mabu, Hirasawa and Hu [51] introduce a technique for the evolution of graph-based candidates, called 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. Li, Yang and Hirasawa [47] extend 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 must be fixed throughout the evolutionary process, meaning that it suffers from the same limitations discussed above.

Poli [65] presents an EC algorithm where solutions are represented directly as graphs. This graph is mapped to a grid, where each row represents a layer of the graph, and columns can be used freely to accommodate the nodes in a layer. By using this representation, it is possible to perform relatively simple mutation and crossover operations to a graph without compromising its syntactic correctness. For the crossover operation, a crossover point (node) is selected in each candidate, and all the ancestors of that node are identified. These two subgraphs of ancestors are then swapped, a process that may overwrite nodes from the other graph but that maintains any original edges. Whenever this swap causes parts of the subgraphs to be placed outside of the grid, these nodes are "wrapped around" their respective rows, meaning that they are moved to the other extreme of the row. For the mutation operator, an existing subgraph is selected and replaced with a newly created one, using the same general mechanism as the crossover. Once again, while this approach makes the implementation of genetic operations simple, it does not allow for correctness constraints (i.e. input-ouput connections) to be maintained. Additionally, this representation assumes that the duplication of nodes is acceptable, which may pose some problems in the domain of Web service composition.

Kantschik and Banzhaf [37] propose linear-graph GP, a structure that allows programs with multiple execution paths to be optmimised using evolutionary computing. This approach is an extension of the simple linear program representation where each element of a

vector contains one program instruction. In this representation, a program graph contains several nodes, and each of these nodes contains a portion of a linear program that may be executed according to preceding branching conditions. Within each of these nodes, there exists also a branching node that is responsible for determining which outgoing edge to follow (i.e. which execution path to choose after executing the current node). The branching functions contained in each of these nodes may read the values of variables manipulated in the linear program portion that is located in the same graph node. The mutation operator in this approach may alter an individual entry within a linear program vector, the branching function within a branching node, or the number of outgoing edges of a given graph node; the crossover operator may exchange individual entries within linear vectors of two candidates, or exchange a group of contiguous graph nodes between two graphs. Despite allowing branching constructs, this approach is not useful to our research because non-sequential relationships between different Web services cannot be encoded into linear representations.

The works in [28, 11, 61] present 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. The reason why these genetic operators can be used without compromising the strucuture of the molecule is that 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.

2.2.2 Other Optimised Composition Approaches

Other methods exist for producing optimised Web service composition results in addition to biologically inspired approaches, and a subset of them is presented in this section. **Tabu search** [29] is a combinatorial optimisation strategy for identifying an optimised solution amongst a group of candidates, typically in problems where exhaustive search is prohibitively expensive. In Tabu search, 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. One of the biggest advantages of Tabu search is that, unlike the hill climbing technique, it can avoid being stuck at local optima when searching. The technique keeps track of a set of tabu moves, which are moves that violate a given collection of tabu conditions. 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). Due to its relative simplicity and its flexibility of implementation, tabu search has a wide range of practical applications for combinatorial optimisation problems. For example, a technique that combines Tabu search and a hybrid Genetic Algorithm (GA) implementation has been proposed recently [62]. For the Tabu search component of the technique, a move is defined as a change in one of the concrete services used as the solution, and the Tabu set is a fixed-size set of the latest n solutions visited. For the hybrid GA, the same basic idea of the traditional GA is applied (set candidate sizes, two-point crossover, mutation that randomly chooses another concrete service), however a local improver is also employed. This improvement process randomly explores a percentage of a

solution's neighbourhood. A drawback of using Tabu search for Web service composition is that the structure used for the representation of candidates is typically linear [62, 5], which restricts the problem model to address semi-automated compositions only.

Integer Linear Programming (ILP) has also been applied to composing Web services [88]. ILP is flexible in the way it represents problems, therefore a fully automated Web service composition approach that also takes non-functional attributes into account when constructing the best solution can be solved using it. An objective function is defined for the achieving the optimal QoS values, and several other functions are used to restrict the functionality of the solutions (i.e. restrict the search space). For linear programming, we first determine the "corners" of the restricted search space (i.e. where two constraint lines meet), and then apply 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 ILP applied to Web service compositions is guaranteed to find an optimal solution, it has been shown to be very inefficient in comparison to EC-based approach as the complexity of the problem increases [3]. Another problem with the ILP approach is that it is likely to pose difficulties when used for creating compositions with multiple execution paths, as branching constraints would have to be represented using a linear function.

Structural Equation Modelling (SEM), a statistical analysis method for forecasting values according to current measurements, has been used in a service selection strategy that considers the future trends of the QoS values of the candidate services [45] as opposed to relying on static QoS snapshots. By being able to predict the QoS trends of services, it is possible to create a composition that is optimal at execution time. One key advantage of SEM is that it is capable of accommodating changes in the user preferences (weights) associated with each QoS attribute, and can use the errors in QoS measurement histories to create a forecast model. The proposed approach was tested using different composition algorithms (e.g. ILP) enhanced with SEM, with a dataset that simulates the changes of QoS parameters over time. Results show that the prediction model for QoS values is initially quite inaccurate, but the accuracy increases for both algorithms as days go by and a history of values is built. While this method is very effective at predicting QoS values to aid in the optimisation of solutions, it cannot be used alone to compose Web services. Thus, SEM is not considered further in this work.

Finally, **Algebraic Expressions (AE)** have been employed to the problem of Web service composition [24, 39]. A formal representation of a Web service composition is used in this approach, relying on algebraic constructs to describe the behaviour of atomic Web services and to constraint 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 [24], involves constructing a mapping between algebraic expression and BPEL, to allow for an automated translation between these two representations. The work in [39] goes further, proposing a composition algorithm that also performs QoS optimisation based on algebraic expressions. Despite promising, the disadvantage of this technique is that it requires the composition task to be described in formal terms that are challenges to system users without the necessary background.

2.3 Multi-objective, Top-K, and Many-objective Composition Approaches

The Web service composition techniques discussed up to this point have the commonality of being single-objective approaches, meaning that they optimise composition solutions based on a one maximising or minimising function. In order to produce a single numerical score while at the same time accounting for multiple QoS attributes, the majority of the objective functions employed by those approaches employs a Simple Additive Weighting (SAW) [34] technique, meaning that all individual QoS attribute scores of a composition are summed after having been scaled by weights that reflect the importance of each quality feature. The objective function used in [18] is a classic example of this SAW technique:

$$fitness_i = w_1 A_i + w_2 R_i + w_3(1 - T_i) + w_4(1 - C_i) \quad (2.1)$$
$$\text{where } \sum_{i=1}^{4} w_i = 1$$

Despite being frequently used in Web service optimisation problems, the SAW technique presents a fundamental limitation: it does not handle the independent and often conflicting nature of the different QoS attributes very well. For example, consider the trade-off between a composition's financial cost and its execution time. Services that have been implemented to respond after very short execution times are likely to be financially more expensive and vice-versa, a trade-off that is not well represented by the SAW model. 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.

Multi-objective (MO) techniques have been extensively employed in QoS-aware Web service composition [49, 92, 89, 87, 86, 13]. The basic idea is to optimise solutions according to a series of independent objective functions that measure the different QoS attributes to be considered. Candidates are then compared based on whether they *dominate* each other, that is, whether the quality scores of one candidate are clearly superior that 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 $(c = 3, t = 1)$ 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 (but 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. Initial approaches to multi-objective Web service composition and selection have focused on employing versions of GA [49], however other EC algorithms have also been considered. The work in [92], for example, proposes the use of a multi-objective Ant Colony Optimization (ACO) algorithm for QoS-aware Web service composition, an algorithm that is particularly suitable to an directed acyclic graph (DAG) composition representation. ACO generally works by building a graph of components (in this case, Web services) that is then 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 pheromones left by other ants. This approach was tested against a multi-objective GA technique using the same fitness function, with results showing that the GA approach converges faster but requires a population 6 times larger than the ACO approach.

HMDPSO, a hybrid multi-objective discrete particle swarm optimisation (PSO) algorithm [87], is another interesting example of MO applied to Web service composition. 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) needs. Each particle is represented as an array that contains concrete candidates, and is divided into multiple parts, each part representing a solution for a different user level. While the paper does not explicitly explain why solutions for multiple user levels are combined in a single particle, it is thought that this is done to reduce the computational expenses associated with MO algorithms. The multi-objective PSO algorithm proposed in this paper updates the positions of particles through the use of crossover operators, and also employs a mutation strategy to prevent stagnation in local optima. The fitness function is responsible for verifying the dominance of a solution over others, and a domination rank is calculated for each solution, with a value of 1 indicating that a solution is non-dominated and a higher values recording the number of dominating others for a given solution. Experiments were conducted to compare the performance of the HMDPSO to that NSGA-II (a GA-based MO algorithm) for the same composition tasks, and results show that HMDPSO with local search is superior to NSGA-II for all objectives and all SLA levels. This is thought to be the case in part because of the more granular fitness function employed by HMDPSO, which differentiates the domination rank of each candidate.

A problem related to multi-objective optimisation is that of selecting the **Top-K** best solutions to a composition problem. According to publications in this area [91], full reliance on multi-objective techniques may not be useful in practice because there are too many possible solutions in the resulting Pareto front, and also because it is difficult for requesters to translate their QoS preferences into weights. Due to this problem, techniques to restrict the number of individuals in the solution set must be applied. Besides using a top-K approach, a ranking of QoS attributes can also be compiled by users to describe the QoS values that are the most important to them [91]. This ranking is used to compute a preference-aware skyline, where a service is only part of the solution set if it is not dominated by another service on a specific number of preferences (this number does not necessarily need to correspond to the full number of QoS attributes considered). The work in [20] is an example of applying a Top-K approach to fully automated composition in large-scale service sets. According to the authors, the main advantage of providing K composition solutions instead of only one is that multiple options are available in case of network failures or similar problems. The issue of large-scale service sets is dealt with by employing a multi-threaded solution that is capable of computing candidates in parallel. Before identifying the Top-K solutions, two preprocessing steps are executed: in the first step, candidate Web services are indexed according to the concepts produced by their outputs (following semantic relationships); in the second step, services that are not useful for the composition task are discarded from consideration using a filtering algorithm. Then, the solutions are identified by employing a MapReduce framework in conjunction with graph planning techniques to compare all potential candidates.

The performance of multi-objective algorithms tends to degrade when optimising solutions according to more than three quality criteria [19], and thus alternative techniques must be investigated to handle such scenarios. These are known as **many-objective techniques**. The work discussed in [19] is based on NSGA-II, an algorithm that allows for the independent optimisation of different quality attributes, and it compares this algorithm to two other techniques: *Maximum Ranking (MR)*, where each solution is ranked for each objective according to its value, from highest to lowest, and the solution's highest ranked objective is used as its overall rank; and *Average Ranking (AR)*, where the ranking values for each objective are computed and added, and each candidate is then ranked according to this

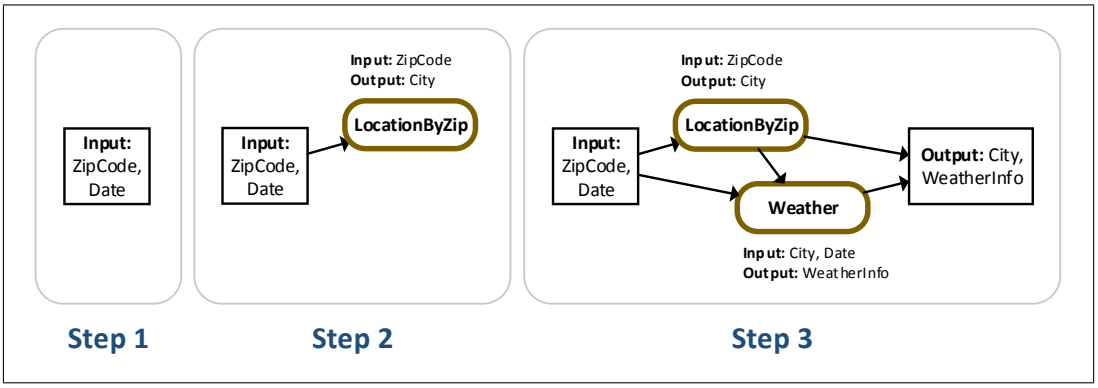


Figure 2.3: Basic example of Web service composition using the Graphplan algorithm.

aggregated value. The experiments conducted using these techniques show that NSGA-II has the worst performance overall for larger composition tasks when considering measurements such as generational distance (i.e. the distance of solutions from the known pareto set), hypervolume (i.e. the are covered by the solutions in the Pareto set), and spread (i.e. the distribution of the solutions within the Pareto front) [4, 36].

2.4 AI Planning-based Composition Approaches

AI planning approaches to Web service composition ensure feasibility by building a composition solution step by step. This solution is represented as a graph, a may either be built to enforce a set of user constraints, or it may be used to find an optimal solution in terms of QoS. A number of works in this area [23, 76, 84, 79] have been based on a fast planning algorithm named Graphplan [8]. 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 has been added that leads to meeting the overall composition objectives (i.e. the composition now produces all of the required outputs). Figure 2.3 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 composition requestor, *ZipCode* and *Date*. In step 2, the service *LocationByZip* is connected, since its input of *ZipCode* can be fulfilled by the existing structure. However, the algorithm continues to execute, 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 [12], authors combine a planning algorithm and a graph search algorithm to achieve both QoS optimisation and feasibility in 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 [73] when performing the backwards traversal, thus finding an optimised solution. The planning graph is extended to include labels associated with each proposition (i.e. 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 determine the optimal solution.

The work in [21] proposes a planning-graph approach to creating a Web service composition technique that is capable of identifying the top K solutions with regards to overall QoS values. According to authors, AI planning is highly suitable to the domain of service composition, however many planning approaches are not efficiently executed. A notable exception to this trend is the planning-graph method, where the search space is greatly reduced through an initial search, thus allowing the remainder of the algorithm to focus on the relevant areas of the search space. In this work, the planning-graph approach is employed in a three-part algorithm. In the first part, a forward search is executed from the input node aiming to find the output node, gradually filtering the services that could be used in the composition. In the second part, the optimal local QoS of each service in the remaining graph is calculated using functions that take into consideration the QoS of the services that could possibly feed its input, also executed from the input node towards the output. Finally, a backward search algorithm is executed to generate the top K solutions according to local values calculated in the previous step (a threshold is provided when running this algorithm to prune out the substandard composition options).

An automated Web service composition approach that uses a filtering algorithm to reduce the number of services considered for the composition, organising the remaining services as a graph according to the ways in which their inputs and outputs match, is proposed in [33]. Once the graph has been determined, a modified dynamic programming approach is applied to it in order to calculate the composition with the optimal QoS. Dynamic programming is a method that breaks problems into smaller subproblems that are then solved, ultimately leading to the solution of the parent problem. In this case, the optimal QoS of each atomic service in the graph is calculated, taking into account its input dependencies. At the end of this process, the overall optimal QoS values are known and the subgraph containing the solution can be extracted by searching the graph backwards. Experiments with the WSC2009 dataset show that the algorithm has good execution times for various dataset sizes, demonstrating its scalability. This work was extended in [35], with the presentation of a composition tool called QSynth, and the performance of formal comparisons with other state-of-the-art approaches, also producing superior results.

A number of other works in the area employ formal AI planning techniques and frameworks to create compositions [7]. The work in [74] presents an approach to include user preferences 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). The syntax of a logic-based language used to specify user preferences is described, allowing for branching according to conditions, and for expressing preferences over 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.

An approach for modelling the data flow between Web services through the use of *domain objects* is presented in [40]. For example, the travel domain contains objects such as flight ticket, hotel reservation, etc. The key idea is to use these objects to connect the composition needs to the services that can address them. In order to do so, authors annotate how each Web service operation relates to a given domain object. By creating this abstraction layer of objects, it is possible to reduce the composition's dependency on implementation details for correct execution. Now services can be thought of as having an object port proxy that leads to specific service ports. Compositions can then be achieved by identifying the necessary domain objects for the required task. The paper goes on to show how this technique can be integrated into existing service composition techniques, in this case

AI-planning based, through the creation of a formal framework. This framework was implemented and run with a virtual travel agency scenario. According to the authors, the implementation of such framework was not trivial, however it successfully demonstrates that service implementations can be modified without impacting the overall composition.

A Web service composition approach that allows users to specify constraints on the data flow of the solutions (i.e. which routes a message is allowed to take and which manipulations it can undergo) is presented in [53]. For example, consider a Web service composition that is supposed to book a holiday for a customer using a flights, accommodation, and a map service. 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 by this approach using a data flow modelling language. This is a visual language that supports the definition of inputs/outputs, forking 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 respecting the constraints of a data net. At the time this paper was written, this approach had not yet been implemented or tested.

2.4.1 Hybrid Approaches

Hybrid approaches combine elements of AI planning and optimisation techniques for solving the composition problem with functionally correct, optimised solutions [15, 68, 85, 14]. These hybrid approaches are quite similar to each other, relying on a directed acyclic graph as the base representation for a candidate solution, and then applying the optimisation techniques to this structure. However, despite incorporating the use of planning techniques, they do not include any discussion on the issue of producing solutions that satisfy user constraints or preferences. Another commonality between these works is that they require the use of SAWSDL-annotated datasets for testing, but these are not widely available to the research community. Therefore, authors developed their own datasets, and utilised each dataset's optimal task solution as the benchmark with which to evaluate the success of their implementation. More specifically, authors calculated the percentage of runs that culminated in the identification of the global optimum as the recommended solution.

In [66], an approach that combines AI planning and an immune-inspired algorithm is used to perform fully automated QoS-aware Web service composition, also considering semantic properties. One significant contribution 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 selects the best solution configuration from a set of candidates. A fitness function considering QoS values and semantic quality is used to judge the best solution, and 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 [67] proposes the employment of the Firefly meta-heuristic technique for performing QoS-aware Web service composition, in conjunction with an AI planning strategy that uses an EPG as the basis for solutions. The firefly meta-heuristic is based on 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. The fitness function takes into account the QoS attributes of the composition.

2.5 Semantic Selection Approaches

One important issue when creating compositions is that of selecting services that are compatible to each other. The simplest way of achieving this is by ensuring that the conceptual output and input types of any two services we wish to connect are perfectly matched, as illustrated by the Graphplan algorithm example in Section 2.4. This involves accessing each service's WSDL, which is a formal description of a Web service's interface [17], and determining whether the two services offer compatible operations. However, in a more realistic scenario it may be very difficult to identify two services with such a precise fit. Because of this, an area of focus in the field of composition is that of semantic Web service selection. The fundamental idea of this approach is to annotate each service with additional semantic information so that the matching of services can be accomplished at a conceptual level. A well-known semantic standard for Web services is OWL-S (Web Ontology Language for Services) [54] standard. OWL-S provides a formal specification of the workings of a service, allows for the association of conceptual classes with each Web service, and supports the use of ontologies that record the relationships between members of these different classes, thus establishing a common vocabulary for inter-service interactions. These features are conducive to automating the handling of Web services, and facilitating the discovery of those that are relevant for a specific task. Another more recent standard for semantic Web service annotation is SAWSDL (Semantic Annotations for WSDL) [43], a language that builds on WSDL by embedding pointers in the WSDL description that refer to semantic concepts. These pointers are called annotations, and are linked to concepts in a higher-level ontology layer.

A number of different selection techniques that use semantic descriptions have been proposed recently [75, 80, 27, 48, 38, 10, 71, 46, 93]. The work in [9], for example, presents a selection strategy that considers more than just WSDL-level descriptions for Web services. In this approach, objects with additional information that is useful to the selection process are associated with each service. These objects have an independent ontology that describes how they interrelate (e.g. a service for making medical appointments has associated objects such as doctor, patient, hospital/clinic, etc), and at composition time the compatibility of these objects is verified. To accomplish this, a framework with service providers, ontology providers, information agents and a composer is proposed. This framework takes selection constraints set by the user into account.

An automated semantic composition method is proposed in [72]. In this method, services are classified according to a functionality tag consisting of an {action, object} pair (e.g. a service for calculating the distance between two cities has the tag {Calculate, Distance}). A service relation graph is created to illustrate the dependencies between concepts, and it is divided in three parts: a graph showing relationships between actions, another graph showing relationships between objects (input/output relationships), and a mapping between items in these graphs. The relationships between these items are determined using domain ontology trees, with the assumption that these trees have already been provided. Given these dependencies, an algorithm is used to find a composition path. The path is found through the action graph and service connections are made based on the object graph, not only the object names. This approach was shown to require substantially less time to execute for larger datasets than previously proposed methods.

The work in [31] proposes a semantic Web service selection method that performs match-

ing based on four distinct levels. At the *Application Domain Matching* level the domain that best matches the user request is identified through the use of category ontologies, and a list of potential service description candidates is retrieved. This is performed by calculating a similarity degree between the user request and the semantic information associated with each domain. Subsequently, at the *Service Description Matching* level, vectors are created for each potential service candidate description based on a given domain ontology, and another vector is created to represent the user requirements. A Vector Space Model is created, employing cosine similarity and TF-IDF to select the best service description. At the *Service Function Matching* level, information from service providers is compared to the service description using a similarity measure, and a set of all services whose functionality fulfils the requirements is returned. Finally, at the *QoS Matching* level, a matching array of quality values showing how closely a Web service matches the user's requirements is calculated, and the optimal candidate is returned.

A framework for performing semantic Web service composition that also allows for user constraints to be specified is proposed in [26]. Initially, services are grouped into distributed "service communities" according to their OWL-S semantic descriptions, where each community has services that cater for similar domains and consequently present similar functionalities. Then, users formulate their composition needs, including the necessary constraints, using terms from the semantic service community descriptions. Effectively, they create an abstract workflow for semi-automated composition by using the community descriptions and specifying their own preferences for the services to be selected. These constraints may be restrictions in input value ranges, in the output value ranges, or other service parameters. A language called KIF was chosen to express the constraints according to the corresponding OWL-S service descriptions. Interestingly, this approach can also handle world-altering services and non-deterministic functionality because it makes use of statecharts to model the behaviour of services.

2.6 Dynamic Web Service Composition

The approaches discussed thus far can be classified as static Web service composition, since they maintain a closed world assumption, that is, they assume that that the quality and the state of the services in the composition remains constant over time. However, in reality the state of the services available in a repository changes as time goes by, causing fluctuations in quality and even occasional failures. The area of dynamic Web service composition removes the closed world assumption, exploring solutions that can cope with quality fluctuations and service failures [1, 6, 82]. The work in [41] proposes an algorithm that quickly creates a solution to satisfy a runtime service request, modelling the service composition problem as a graph in which a path is to be found. Forward and backward chaining approaches based on input/output matches are used for path exploration, relying on heuristics to encourage the exploration of the most promising paths and to reduce the number of services considered. In the approach proposed by this paper, both forward chaining and backward chaining are employed simultaneously, with the intention of having their paths meet in the middle. By doing so, the number of branches to be considered is greatly reduced. Experiments showed that the bidirectional search requires the exploration of a consistently smaller number of services when compared to the exclusively forward and exclusively backward approaches.

A multi-objective Web service selection approach that optimises solutions according to two functions, a static one that calculates the overall quality of service (QoS) of a solution, and a dynamic one which calculates the *trust degree* of a solution at a given time, is presented in [77]. The trust degree is defined as the number of successful executions of a service over

its total number of executions, a piece of information that can be obtained by analysing execution logs. The optimisation algorithm used in this approach is an adaptive form of ant colony optimisation (ACO), where pheromones are adjusted according to the trust degree (calculated anew at every iteration) and the QoS is used by each ant as the heuristic for choosing the next node to visit. Each node in the graph explored by the ants represents an abstract service, with multiple concrete service candidates which can provide that functionality. The algorithm works by generating a set of of solutions and then identifying its Pareto subset by comparing all solutions. The Pareto subset is used in the next iteration, for updating the path pheromones. A case study is presented, comparing the performance of adaptive ACO to that of of the standard ACO algorithm, with results showing that adaptive ACO has a higher accuracy percentage than the standard ACO.

The notion of transactions can be associated with service compositions, where different transactional properties guarantee certain runtime behaviours in a dynamic environment. The work in [22] proposes a semi-automated Web service composition approach that not only takes the functionality and quality of the services into account, but also their transactional properties. In this work, transaction properties are defined as the behaviour of Web services when interacting with one another. Knowing the behaviour of Web services is important for estimating how reliable their execution is and which ones might require recovery strategies at runtime. A service is considered *retriable* if it can terminate successfully after multiple invocations, *compensatable* if there is another service that can semantically undo its effects, and *pivot* if its effects cannot be undone once it is executed but if it fails there are no effects. The system takes an abstract workflow and a set of user preferences as its inputs, where the user preferences contain QoS weights and risk levels corresponding to the transitional requirements for the composition. Then, a planner engine assigns one concrete Web service to each abstract slot of the provided workflow. Whenever a service is assigned to a part of the workflow, its transactional properties influence any subsequent services, thus the risk must be recalculated along with the overall QoS. Experiments were run for various risk scenarios, and computation time was found to remain under 2 seconds even for the largest dataset (comprising 3602 atomic Web services), at the same time meeting user preferences.

An approach to QoS-aware Web service composition that is focused on *rebinding*, that is, deciding which concrete services to bind to each abstract task at runtime in order to take the current state of the environment into consideration, is presented in [63]. This approach considers global QoS constraints (e.g. the overall composition price must be lower than 5), local QoS constraints (e.g. the individual price for a service must be no higher than 1), and service dependency constraints (e.g. as many services as possible should be used from the same provider). Two techniques are employed during the composition process: GRASP, which is used to construct initial solutions, and Path Relinking, which is used to perform improvement on these solutions. *GRASP* (Greedy Randomized Adaptive Search Procedure) iteratively builds a *valid* solution vector, adding one candidate to fulfil each solution slot at a time and ensuring that this candidate respects the pre-established user constraints. The order of candidate addition matters, so it is performed randomly each time. Once a valid solution has been built, GRASP identifies a list of replacement candidates for each task slot, including the most promising candidates while also respecting the constraints observed by the valid solution. *Path Relinking* explores the neighbouring solutions of the initial valid solution, seeking to further improve it. To do so, it slowly modifies solutions by changing services from one task slot at a time. The objective function used in this paper encourages the improvement of QoS values at the same time it enforces the relevant user constraints.

2.7 Summary and Limitations

This chapter presented an overview of the recent research conducted on different aspects of the automated Web service composition problem. The first area explored was that of **single-objective composition**, which aims to create composite solutions with the best possible overall Quality of Service (QoS) by conducting optimisations according to an objective function. Different approaches have been attempted for this, including both traditional approaches such as Integer Linear Programming (ILP) and Evolutionary Computation (EC) approaches such as Genetic Programming (GP), though in certain cases traditional approaches may not scale well as the composition problems grow in complexity. One key limitation of single-objective composition approaches is that they neglect the issue of branching, that is, they do not allow the creation of solutions with multiple alternative outputs that may be produced depending on a condition. To overcome this problem, a new candidate representation that also encodes this form of conditional branching needs to be proposed.

Another area discussed is that of **multi-objective, top-K, and many-objective composition**, where each QoS attribute is optimised separately, and a set of solutions presenting trade-offs between these different quality attributes is generated. Multi-objective approaches work best for optimisation problems that involve two or three independent dimensions, while many-objective approaches are capable of handling more than three of them; top-K aims to produce a predetermined number of solution options based on a ranking strategy. The optimisation of multiple independent objectives is complex and provides many possibilities for further exploration. One interesting problem is that of many-objective optimisation for SLA-aware Web service composition, which refers to the constrained optimisation of several independent objectives to reflect the minimum quality requirements of the composition requestor. Accomplishing this is particularly challenging when creating solutions with multiple output possibilities.

A number of **AI planning-based composition** approaches are also presented in this chapter. Their fundamental idea is to build solutions step by step, adding one atomic service to the composition at a time and subsequently checking whether the overall desired output has now been produced. These approaches are conducive to ensuring that the generated solutions are feasible and also included conditional branches into the solution's workflow whenever necessary. Despite these advantages, it is difficult to globally optimise the quality of a solution produced through planning, since its components are not modified or improved once they have been selected. A promising strategy to overcome this problem would be to combine planning algorithms with a population-based approach for optimisation, though this avenue has not yet been significantly explored by researchers.

The **semantic Web service selection** approaches examined in this chapter propose a more realistic way of selecting the atomic components of a composition. Instead of expecting the return values and parameter types of service operations to match perfectly, the closest possible output-input matches are calculated using semantic distance measurements. The limitation of most works in this area is that they restrict their selection technique to semi-automated composition, where it is assumed that a framework with abstract service slots that are to be filled by concrete atomic services has already been provided. In the case of composition approaches based on AI planning techniques, where the workflow is built as services are connected to the solution, more flexible semantic selection methods are necessary. This motivates further research in this area.

Finally, this chapter focuses on the issue of **dynamic Web service composition**. In this type of composition a closed world is abandoned, meaning that the state of services is expected to change throughout time. This brings two main problems: firstly, when the quality of the services in a repository changes, a solution that was once optimised according

to the previous quality measures may suddenly transform into a low-standard alternative; secondly, certain services may become unavailable, meaning that solutions which incorporate them are henceforth unusable. The dynamic composition approaches discussed in this chapter use a variety of self-healing and recovery techniques to adjust solutions according to these changes, however they do not rely on EC techniques for doing so. These techniques can quickly (re)optimise the QoS of existing solutions and provide composition backups (i.e. other candidates in the population) in case of failure, thus making EC an interesting dynamic alternative.

Chapter 3

Preliminary Work

3.1 Problem Formalisation

A typical Web service composition scenario is that of an online book shopping system, as shown in Figure 3.1. The objective of this system, constructed using existing Web services, is to allow a customer to purchase a book offering different methods of payment according to their account balance. Namely, if the customer has enough money to pay for the selected book, then they would like to pay for it in full; otherwise, they would like to pay for it in instalments. The idea is to construct this system in a fully automated manner, meaning that the services to be used, the connections between these services, and the overall flow of the system are determined automatically. When requesting the creation of a system with the desired functionality, the composition task provided consists of the overall inputs required by the system (e.g. book title, author, customer information), conditional constraints (e.g. the customer's account balance is less than the price of the chosen book), and the potential outputs produced by the system (e.g. a receipt, if the customer paid in full, or an initial bill, if the customer is paying in instalments).

More formally, a Web service composition that includes branching can be thought of as a reachability problem where, given a composition input set I, a group of services is assembled in order to produce multiple composition output sets O_1 to O_m, each output being dependent on a sequence of branching conditions C_1 to C_n that leads to that outcome. Users often expect these services to be optimised according to their quality, giving higher weights to reflect the priority of quality attributes. Based on this understanding, a *composition task* can be represented as a tuple $T = (P, W)$. P is a collection of paths from the provided input I, going through some conditions C to reach one of required output possibilities O_i, essentially forming a tree $P = \{\langle I, C, O_i \rangle\}$, and W is the set of priority weights for each Quality of Service (QoS) attribute. A *Web service* is a tuple $S_x = (I_x, O_x, Q_x)$, where I_x is the set of inputs that must be provided in order to execute the service, O_x is the set of outputs produced by the service after execution, and Q_x are the service's QoS attributes. Finally, a *Web service composition* is a minimal set of paths where all services appearing in a path are sequentially linked, and where interleaved conditional nodes are allowed. Thus, it can be described as $WSC = \{\langle I, (S, C_x)\star, S, O_i \rangle\}$. The inputs of all services in the sequences S are fully satisfied, if all paths in the composition are considered, and the same is true for the various O possibilities. A property of the paths in WSC is that if all services are removed from a path $a \in WSC$, then its shortened version corresponds to a path in P (i.e. $a_{short} \in P$).

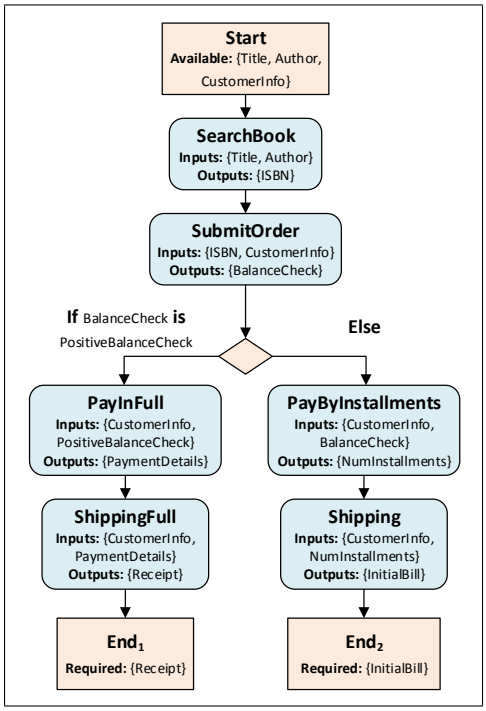


Figure 3.1: A Web composition for an online book shopping system.

3.2 Language Constructs and QoS

In addition to the functional aspects of Web service composition, the goodness of the services included in a solution also plays a part in the creation of a composite system. This non-functional set of attributes is known as Quality of Service (QoS) [56], and it measures characteristics that are desirable in a service from a customer's point of view. In this work, four QoS attributes are considered [90]: 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, and Reliability (R), which is the likelihood of a service responding appropriately when invoked. The existing languages for Web service composition (e.g. BPEL4WS [83]) use certain constructs to control the flow of the resulting systems 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 3.2. The total cost and time of this construct are calculated by adding the values 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 3.3. 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. This

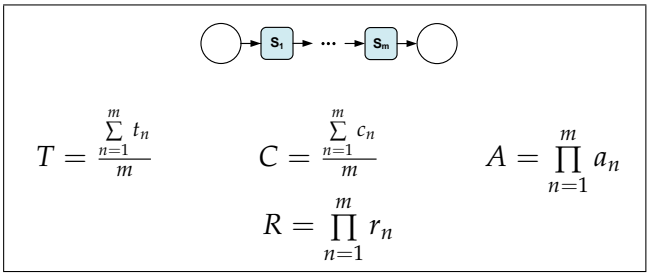


Figure 3.2: Sequence construct and calculation of its QoS properties.

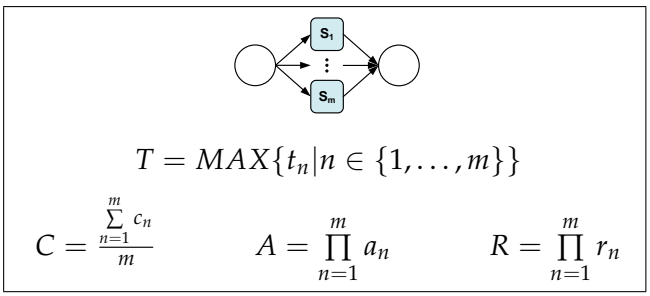


Figure 3.3: Parallel construct and calculation of its QoS properties.

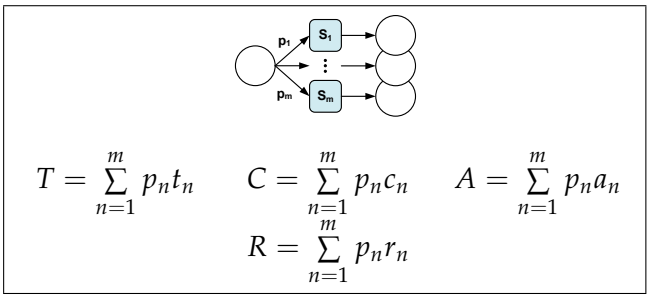


Figure 3.4: Choice construct and calculation of its QoS properties.

is shown in Figure 3.4. In this construct, all overall QoS attributes are calculated as a weighted sum of the services from each individual path, where each weight corresponds to probability of that path being chosen during runtime. These weights add up to 1.

3.3 Fitness Function

The fitness function is used for optimising solutions according to their overall Quality of Service (QoS), and was based on the function shown in [**?**]. This function measures the overall quality of a composition candidate by performing a weighted sum of the overall QoS attributes of a given candidate:

$$fitness_i = w_1 A_i + w_2 R_i + w_3(1 - T_i) + w_4(1 - C_i) \tag{3.1}$$

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

This function produces values in the range [0,1], where a fitness of 1 means the best possible quality, and a fitness of 0 means the worst. Because this is a maximising function,

the Time T and cost C are offset by 1 in the formula, so that higher scores correspond to better qualities for these attributes as well. The overall quality attributes A, R, T, and C of a composition are calculated according to formulae shown in Figures 3.2, 3.3, and 3.4, and these values are then normalised to fit within the [0,1] range, according to Formulae 3.2, 3.3, 3.4, and 3.5. These formulae are inspired by the work in [**?**], and normalise the original overall composition values (A_{orig}, R_{orig}, T_{orig}, C_{orig}) by using the overall minimum and maximum values for each quality attribute over the entire service repository. In the formulae for calculating T_i and C_i the maximum values are multiplied by the total number n of services in the repository, thus creating an upper bound for the normalisation. This is because a composition including all available candidate services would have the worst possible time and cost attributes.

$$A_i = \begin{cases} \frac{A_{orig} - A_{min}}{A_{max} - A_{min}}, & \text{if } A_{max} - A_{min} \neq 0. \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

$$R_i = \begin{cases} \frac{R_{orig} - R_{min}}{R_{max} - R_{min}}, & \text{if } R_{max} - R_{min} \neq 0. \\ 1 & \text{otherwise.} \end{cases} \tag{3.3}$$

$$T_i = \begin{cases} \frac{T_{orig} - T_{min}}{nT_{max} - T_{min}}, & \text{if } nT_{max} - T_{min} \neq 0. \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

$$C_i = \begin{cases} \frac{C_{orig} - C_{min}}{nC_{max} - C_{min}}, & \text{if } nC_{max} - C_{min} \neq 0. \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

3.4 Conditional Tree Representation

3.4.1 Movitation

3.4.2 Proposed Approach

Our proposed approach employs Genetic Programming to evolve solutions according to their overall Quality of Service, meanwhile maintaining their functional correctness. A candidate solution for a composition is represented as a tree, where the non-terminal nodes represent the composition flow constructs (sequence, parallel, and conditional), and the terminal (leaf) nodes represent the atomic Web services included in the composition. An example of such a tree is shown in Figure 3.5. To calculate the overall QoS of a tree, the formulae in Figures 3.2, 3.3 and 3.4 are employed, where the children of a current node act as the services used in the calculations. For Figure 3.5, for example, the QoS is first calculated for the sequence nodes directly above the leaves, then for the conditional node, and finally for the root node. This is discussed further in Subsection **??**.

A population initialisation algorithm similar to the one in [76] is employed, and so are constrained mutation and crossover operations. However, the difference of the current approach in comparison to [76] is that it also considers the case where a user requires a branching constraint to be met, like the one illustrated in Figure 3.1. The presence of branching means that different values are produced by a service at runtime, and these concrete values are subtypes of a statically defined concept. For example, a service that statically outputs a *fruit* concept may output the subtype *banana* at runtime, and thus branching conditions may be defined based on the different kinds of fruits. The relationship between types is held by a taxonomy that encodes the inputs and outputs pertinent to the candidate atomic services employed in the composition. For simplicity, the implementation presented in this

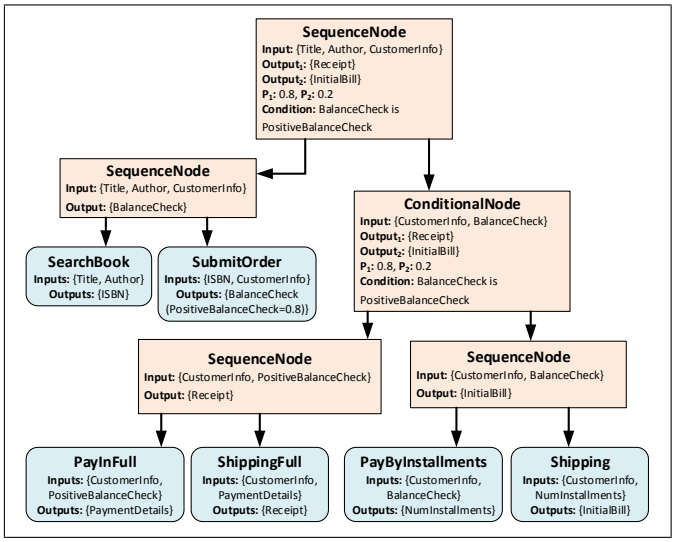


Figure 3.5: Example of tree representation for Web service composition.

work handles branching with two options (if and else), so this scenario will be the focus throughout the following sections.

Population Initialisation

As opposed to generating a composition candidate purely based on a set of available inputs and another of desired outputs, when handling branching constraints it is also necessary to consider a branching condition. Thus, a composition task with branching must include a set of available inputs, a condition of the is-a kind (e.g. if $banana \succeq fruit$), and two sets of outputs, one expected in case the condition is met and the other in case it is not. With this information it is possible to run the initialisation algorithm and create a composition candidate in a graph format, later translated into a tree representation.

Algorithm 4 is used to create a candidate which may incorporate a branching constraint, though it is also capable of handling unconditional tasks. It requires a set of inputs I, an if-else condition C, two sets of outputs O_1 (for if-branch) and O_2 (for else-branch). The set of probabilities P is only required for a specific case, explained below, and C as well as O_2 are not required for tasks without branching. The intuition behind this algorithm is to assemble a candidate tree in parts. Firstly, if the task has in fact two sets of outputs and a condition C, then two sub-composition graphs G_1 and G_2 are generated to represent the if and the else branches, respectively. This generation is performed using the *createGraph* algorithm proposed in [76]. Subsequently, these graphs are translated to trees T_1 and T_2 using the *toTree* procedure proposed in Algorithm 2, and included as children of a *ConditionalNode* created for condition C (the left child represent the if-branch, and the right child the else-branch).

Secondly, the algorithm verifies whether the value used for condition C can be met by using the set of inputs I. If it can, then the set of probabilities P of each branch being executed is associated to the *ConditionalNode* and the tree is returned (this assumes that the probabilities for obtaining a specific value for the if condition in C are already known – typically during mutation). If it cannot, then another sub-composition graph G_4 is generated and translated to T_4, creating the part of the composition that leads from the available inputs I to the value used in condition C. In this case, the probability associated with each branch of the condition is extracted from the last service in T_4 (i.e. the service that produces the overall

Input : I, O_1, O_2, C, P
Output: candidate tree T

if $O_2 \neq \emptyset$ **then**
  $G_1 \leftarrow \texttt{createGraph}(I \cup C.if, O_1)$;
  $G_2 \leftarrow \texttt{createGraph}(I \cup C.else, O_2)$;
  $T_1 \leftarrow \texttt{toTree}(G_1.input)$;
  $T_2 \leftarrow \texttt{toTree}(G_2.input)$;
  $T_3 \leftarrow$ new $ConditionalNode(C)$;
  $T_3.leftChild \leftarrow T_1$;
  $T_3.rightChild \leftarrow T_2$;
  if $C \sqsubseteq I$ **then**
    $T_3.prob \leftarrow P$;
    return T_3;
  else
    $G_4 \leftarrow \texttt{createGraph}(I, C.else)$;
    $T_4 \leftarrow \texttt{toTree}(G_4.input)$;
    $T_3.prob \leftarrow T_4.final.P$;
    $T \leftarrow$ new $SequenceNode()$;
    $T.leftChild \leftarrow T_4$;
    $T.rightChild \leftarrow T_3$;
    return T;
  end
else
  $G \leftarrow \texttt{createGraph}(I, O_1)$;
  $T \leftarrow \texttt{toTree}(G.input)$;
  return T;
end

ALGORITHM 1. Generating a new candidate tree or a mutated subtree.

```
Procedure toTree()
    Input  : N
    Output: tree T
1:  if N is leaf then
2:      return N;
3:  else if N = input then
4:      if |N.to| = 1 then
5:          T ← toTree(next.to[0]);
6:      else
7:          T ← createParallelNode(N.to);
8:      end
9:  else
10:     r;
11:     children ← N.to − output;
12:     if |children| = 1 then
13:         r ← toTree(children[0]);
14:     else
15:         r ← createParallelNode(children);
16:     end
17:     T ← new SequenceNode();
18:     T.leftChild ← N;
19:     T.rightChild ← r;
20: return T;
Procedure createParallelNode()
    Input  : children
    Output: tree T
21: T ← new ParallelNode();
22: subtrees ← {};
23: foreach c in children do
24:     S ← toTree(c);
25:     subtrees ← subtrees ∪ {S};
26: end
27: T.children ← subtrees;
28: return T;
```

ALGORITHM 2. Converting graph into tree representation.

sub-composition output that satisfies the value used in condition C), and a *SequenceNode* is created as the tree root to link this deterministic part of the composition (T_4) to the conditional part (T_3). Finally, if no condition C and output set O_2 are provided, then the candidate is generated purely by using the *createGraph* and *toTree* algorithms.

For reasons of brevity, the *toTree* procedure shown in Algorithm 2 for converting a graph representation of a composition to a tree representation is not explained in detail. Its general idea is the same as the one discussed in [**?**], which is to recursively traverse a graph, starting from the start ($input$) node and working towards the end ($output$) node. At each node, the number of outgoing edges determines whether to create a sequential or a parallel construct, and *toTree* recurses on each of the destination nodes of these outgoing edges. It is important to note that after creating the candidate tree, it must be traversed to determine the input values required to execute each node of the tree, and the output values produced by each node. This information is recorded within each node.

Mutation and Crossover

The crossover operator employed in the evolutionary process swaps any two leaf nodes (i.e. Web services), one from each candidate, provided that these two leaves are functionally equivalent in terms of input and output values but represent different Web services. This particular crossover operation implementation was chosen because it provides an effective mechanism for performing Web service selection, whose objective is to identify the best service to perform a task out of a set of candidates providing the same functionality. The mutation operator selects a node of the candidate tree at random, and using its input and output information produces a new subtree that may be structurally different yet provides the same functionality. The subtree is generated using Algorithm 4, and is used to replace the originally selected node in the candidate.

GP Algorithm for Composition

When applying GP to the service composition problem, each candidate in the population represents a possible candidate solution. The initial population is created as described above, and the fitness of each candidate is evaluated using the fitness function. The fittest individuals are reproduced to the next generation, while other candidates are created in the next generation by applying the mutation and crossover operations to currently promising individuals. This new generation is then evaluated according to the fitness of candidates, and the process is repeated for a fixed number of generations. Lastly, the candidate with the highest fitness in the final population is chosen as the composition solution.

3.5 Conditional Graph Representation

3.5.1 Why Add Branches

3.5.2 Proposed Representation

The approach proposed in this work is an extension of the GraphEvol technique [**?**] to allow for the representation of composite services with multiple execution branches, depending on conditional constraints specified in the composition task. A GraphEvol candidate may look like the representation shown in Figure 3.1, with each block encoded as a graph node, and edges flowing from the input (start) towards the output (end) nodes. However, nodes with multiple children and/or multiple parents can also be included when necessary, meaning

that Directed Acyclic Graphs (DAGs) are the basis of this representation. Algorithm 3 describes the general procedure followed by GraphEvol, and the following subsections explain the ways in which this basic technique was extended to allow for conditional branching.

1. Initialise the population using the graph building algorithm.
2. Evaluate the fitness of the initialised population.
while *max. generations not met* **do**
  3. Select the fittest graph candidates for reproduction.
  4. Perform mutation and crossover on the selected candidates, generating offspring.
  5. Evaluate the fitness of the new graph individuals.
  6. Replace the lowest-fitness individuals in the population with the new graph individuals.
end

ALGORITHM 3. Steps of the GraphEvol technique [**?**].

Graph building algorithm

As with the fundamental GraphEvol technique, a graph-building algorithm is used to initialise the population of candidates so that the connections between the different atomic services included in a composition are correct. In this work, however, this algorithm is extended to create solutions with one input but multiple outputs, depending on the branching conditions specified. This extended version is shown in Algorithm 4`null`.

Candidates are created using the *buildGraph* procedure. This procedure requires $InputNode$, which is the root of the composition's task tree, a list of *relevant* candidate services from the service repository, and optionally a *candMap*. The *candMap* is used for the crossover procedure only, so it will be discussed later. Given these inputs, the algorithm proceeds to connect nodes to the graph, one at a time, until a complete solution is found. As explained earlier, the resulting composition will have several independent branches, thus the recursive procedure *buildBranch* has been created to handle each part of the composition. After connecting the *start* node to the graph, we execute *buildBranch* providing the first task it should achieve (i.e. *TaskNode*, which initially will be a conditional branching node), a list of candidates *candList* that contains services that are executable/reachable using the *start* node outputs, the partially built graph G, and other relevant data structures. Once the *buildBranch* procedure has finished executing, the graph G will be completed. The algorithm used for construction creates graphs from the *start* node to the *end* nodes in order to prevent cycles from forming, but this may lead to *dangling* nodes, which are nodes that do not have any outgoing edges despite not being *end* nodes. These are redundant parts of the solution, and thus they must be removed once G is built. Finally, the creation of the new candidate graph is finished.

Algorithm 4`null` also describes the *connectNode* procedure, 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 procedure 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 both the values required by the *if* case and by the *else* case when using the service we have just connected. On the other hand, if the *TaskNode* represents the *end* of a branch, we check that the list $allInputs$ of potential inputs contain all values necessary to satisfy the inputs fo the *end* node.

Algorithm 5`null` 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

Procedure `buildGraph`$(InputNode, relevant, candMap)$

$start.outputs \leftarrow \{InputNode.values\}$;
$TaskNode \leftarrow InputNode.child$;
$G.edges \leftarrow \{\}$;
$G.nodes \leftarrow \{\}$;
$allInputs \leftarrow \{\}$;
$connections \leftarrow \{\}$;
`connectNode`$(start, connections, G, allInputs, TaskNode)$;
$allowedAncestors \leftarrow \{start\}$;
if *candMap is* `null` **then**
  $candList \leftarrow$ `findCands`$(start, allowedAncestors, relevant)$;
else
  $candList \leftarrow \{node | (start, node) \in candMap\}$;
end
`buildBranch`$(TaskNode, candList, connections, allInputs, G,$
$relevant, allowedAncestors, candMap)$;
`removeDangling`(G);
return G;

Procedure `connectNode`$(n, connections, G, allInputs, TaskNode)$

$n.objective \leftarrow TaskNode$;
$G.nodes \leftarrow G.nodes \cup \{n\}$;
$G.edges \leftarrow G.edges \cup connections$;
if *TaskNode is ConditionNode* **then**
  if $|n.outputs| > 1$ **then**
    return $((TaskNode.general \sqsubseteq n.outputs.general \wedge TaskNode.specific \sqsubseteq$
    $n.outputs.specific), n)$;
  else
    return $(\texttt{false}, n)$;
  end
else
  return $(TaskNode.outputs \sqsubseteq allInputs, n)$;
end

ALGORITHM 4. Procedures for building a new candidate graph and for connecting a particular node to the graph [**?**].

```
Procedure buildBranch(TaskNode, candList, allInputs, G,
relevant, allowedAncestors, candMap)
    goalReached ← false;
    connResult;
    while ¬goalReached do
        found ← false;
        foreach cand ∈ candList do
            connections ← {};
            ancestors ← {x.outputs | x ∈ G ∧ x ∈ allowedAncestors};
            if cand.inputs ⊑ ancestors then
                connections ← connections ∪ {x ← minimal(ancestors)};
                found ← true;
            end
            if found then
                connResult ←
                connectNode(cand, connections, G, allInputs, TaskNode);
                goalReached ← connResult[0];
                allowedAncestors ← allowedAncestors ∪ {cand};
                if candMap is null then
                    candList ← candList∪
                    findCands(cand, allowedAncestors, relevant);
                else
                    candList ← candList ∪ {node | (cand, node) ∈ candMap};
                end
                break;
            end
        end
        candList ← candList − {cand}
    end
    connectTaskNode(TaskNode, connResult, G, allowedAncestors,
    candList, candMap);
```

ALGORITHM 5. Indirectly recursive procedure for building one of the branches of the new candidate graph.

candList are considered for addition. A candidate *cand* is randomly chosen from *candList*, and it is connected to the graph (using the *connectNode*) procedure if all of its inputs can be fulfilled by the *ancestor* outputs (i.e. the outputs of nodes already present in that particular execution branch). The set of services in *connections*, which are used to connect *cand* to *G*, is a minimal set, meaning that the output of these services fulfils all the inputs of *cand*, but if any connection is removed from the set that is no longer the case. After each *cand* service is connected to *G*, the *candList* 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 construction of that branch, either by connecting an *end* node to it or by further splitting the branch according to a new *TaskNode* condition. In case of the latter, *connectTaskNode* will invoke the *buildBranch* procedure again.

Procedure `connectTaskNode`(*TaskNode*, *connResult*, *G*, *allowedAncestors*, *candList*, *candMap*)
  if *TaskNode is ConditionalNode* **then**
    $TaskGoal.probs \leftarrow connResult[1].probs$;
    $G.nodes \leftarrow G.nodes \cup \{TaskNode\}$;
    $G.edges \leftarrow G.edges \cup \{connResult[1] \rightarrow TaskNode\}$;
    $allowedAncestors \leftarrow allowedAncestors \cup \{TaskNode\}$;
    $connections \leftarrow \{\}$;
    if *candMap is* `null` **then**
      $candList \leftarrow candList \cup$
      `findCands`(*TaskNode*, *allowedAncestors*, *relevant*);
    else
      $candList \leftarrow candList \cup \{node | (TaskNode, node) \in candMap\}$;
    end
    $allInputs \leftarrow \{\}$;
    $ifChild \leftarrow TaskNode.ifChild$;
    if *ifChild is OutputNode* **then**
      $allInputs \leftarrow \{x.outputs | x \in allowedAncestors\}$;
    end
    `buildBranch`(*ifChild*, *candList*, *allInputs*, *G*, *relevant*, *allowedAncestors*, *candMap*);
    $elseChild \leftarrow TaskNode.elseChild$;
    if *elseChild is OutputNode* **then**
      $allInputs \leftarrow \{x.outputs | x \in allowedAncestors\}$;
    end
    `buildBranch`(*elseChild*, *candList*, *allInputs*, *G*, *relevant*, *allowedAncestors*, *candMap*);
  else
    $ancestors \leftarrow \{x.outputs | x in G \wedge x \in allowedAncestors\}$;
    $connections \leftarrow \{x \rightarrow TaskNode \, | x \in minimal(ancestors)\}$;
    $G.nodes \leftarrow G.nodes \cup \{TaskNode\}$;
    $G.edges \leftarrow G.edges \cup connections$;
  end

ALGORITHM 6. Procedure for finishing construction of a branch, splitting it further in case another condition exists.

As previously explained, Algorithm 6`null` is responsible for finishing the construction

of a given execution branch, according to one of two scenarios. In the first scenario, the *TaskNode* reached is a conditional node, meaning that the branch will be further split into an *if-and-else* structure. In this case, the *TaskNode* is added to G, connected through the previously added service in $connResult[1]$ (i.e. the service that fulfilled the outputs required for the condition to be checked). Whenever a branching node is added, probability values must be associated with each path, indicating the likelihood of that path being executed. Since the branching occurs based on the values produced by $connResult[1]$, the probabilities of producing these different output possibilities are copied from this service. Then, 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 services already in the graph which produce all the outputs required by this *end* node.

Mutation and Crossover

Procedure `mutation`$(G, InputNode, relevant)$
  $n \leftarrow$ `selectNode`(G);
  if *n is start* **then**
    return `buildGraph`$(InputNode, relevant,$ `null`$)$;
  else
    $TaskNode \leftarrow n.objective$;
    `removeNodes`(n);
    $allInputs \leftarrow \{\}$;
    $candList \leftarrow \{\}$;
    $allowedAncestors \leftarrow \{\}$;
    foreach $node \in G.nodes$ **do**
      $allowedAncestors \leftarrow allowedAncestors \cup \{node\}$;
    end
    foreach $node \in G.nodes$ **do**
      $candList \leftarrow candList \cup$ `findCands`$(node, allowedAncestors, relevant)$;
    end
    if *TaskNode is OutputNode* **then**
      $allInputs \leftarrow \{x.outputs | x \in allowedAncestors\}$;
    end
    return `buildBranch`$(TaskNode, candList, allInputs, G, relevant,$
    $allowedAncestors,$`null`$)$;
  end
Procedure `crossover`$(G_1, G_2, InputNode, relevant)$
  $candMap \leftarrow \{(x,y) | x \rightarrow y \in G_1.edges\}$;
  $candMap \leftarrow candMap \cup \{(x,y) | x \rightarrow y \in G_2.edges\}$;
  return `buildGraph`$(InputNode, relevant, candMap)$;

ALGORITHM 7. Procedures for performing mutation and crossover on graph candidates [**?**].

The procedures for performing the mutation and crossover operations are shown in Algorithm 7`null`. 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* or a *condition* node.

If this node is the *start* node, an entirely new candidate graph is constructed; otherwise, all nodes whose input satisfaction depends upon node n are removed from G, and so are any subsequent splits of that branch. The construction of this partially-built graph is then finished by invoking the *buildBranch* procedure and providing the original *TaskNode* (n's objective) and appropriate data structures to it. The *allowedAncestors*, the *candList*, and *allInputs* are calculated based on the remaining nodes of G. The mutation operator was designed in this way so that it allows for variations of the original candidate, at the same time maintaining the correctness of the connections between services (nodes) in the graph.

In the case of *crossover*, the general idea is to reuse connection patterns from two existing candidates G_1 and G_2 in order to create a new child candidate that combines elements from these two parents. In order to do so, the original connections of G_1 and G_2 are abstracted into a map called *candMap*. This map can be queried to determine all the connections (from both parents) that can be made starting from a given node x, e.g. x connects to y_1, y_2, and y_3 in G_1 and G_2. After having assembled this map, the *buildGraph* procedure is invoked to create a child candidate. The difference is that the addition of candidates to the *candList* is done by querying the *candMap* to determine which services could be reached from the current node according to the connection patterns in the original parents. One of the advantages of this crossover implementation is that it allows for a flexible operation that reuses connection information from both parents. Additionally, this operation can be executed using the already existing graph-building algorithm with minimal changes.

3.6 Experiments

Chapter 4

Proposed Contributions and Project Plan

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] BADER, J. M. *Hypervolume-based search for multiobjective optimization: theory and methods*. Johannes Bader, 2010.

[5] 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.

[6] BERG, H. V. D., ET AL. Revenue optimization of service compositions using conditional request retries. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 1–9.

[7] BERTOLI, P., KAZHAMIAKIN, R., PAOLUCCI, M., PISTORE, M., RAIK, H., AND WAGNER, M. Control flow requirements for automated service composition. In *Web Services, 2009. ICWS 2009. IEEE International Conference on* (2009), IEEE, pp. 17–24.

[8] BLUM, A. L., AND FURST, M. L. Fast planning through planning graph analysis. *Artificial intelligence 90*, 1 (1997), 281–300.

[9] 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.

[10] 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.

[11] 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.

[12] CHEN, M., AND YAN, Y. Qos-aware service composition over graphplan through graph reachability. In *Services Computing (SCC), 2014 IEEE International Conference on* (2014), IEEE, pp. 544–551.

[13] 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.

[14] 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.

[15] COTTA, C., AND FERNÁNDEZ, A. J. Memetic algorithms in planning, scheduling, and timetabling. In *Evolutionary Scheduling*. Springer, 2007, pp. 1–30.

[16] CRAENEN, B., EIBEN, A., AND MARCHIORI, E. How to handle constraints with evolutionary algorithms. *Practical Handbook Of Genetic Algorithms: Applications* (2001), 341–361.

[17] 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.

[18] DA SILVA, A. S., MA, H., AND ZHANG, M. A graph-based particle swarm optimisation approach to qos-aware web service composition and selection. In *Evolutionary Computation (CEC), 2014 IEEE Congress on* (2014), IEEE, pp. 3127–3134.

[19] DE CAMPOS, A., POZO, A. T., VERGILIO, S. R., AND SAVEGNAGO, T. Many-objective evolutionary algorithms in the composition of web services. In *Neural Networks (SBRN), 2010 Eleventh Brazilian Symposium on* (2010), IEEE, pp. 152–157.

[20] DENG, S., HUANG, L., TAN, W., AND WU, Z. Top- k automatic service composition: A parallel method for large-scale service sets. *Automation Science and Engineering, IEEE Transactions on 11*, 3 (July 2014), 891–905.

[21] 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.

[22] EL HADAD, J., MANOUVRIER, M., AND RUKOZ, M. Tqos: Transactional and qos-aware selection algorithm for automatic web service composition. *Services Computing, IEEE Transactions on 3*, 1 (2010), 73–85.

[23] FENG, Y., NGAN, L. D., AND KANAGASABAI, R. Dynamic service composition with service-dependent qos attributes. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 10–17.

[24] FERRARA, A. Web services: a process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing* (2004), ACM, pp. 242–251.

[25] GALVAN-LOPEZ, E. Efficient graph-based genetic programming representation with multiple outputs. *International Journal of Automation and Computing 5*, 1 (2008), 81–89.

[26] 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 *Web Services, 2008. ICWS'08. IEEE International Conference on* (2008), IEEE, pp. 228–237.

[27] GARCÍA, J. M., RUIZ, D., RUIZ-CORTÉS, A., AND PAREJO, J. A. Qos-aware semantic service selection: An optimization problem. In *Services-Part I, 2008. IEEE Congress on* (2008), IEEE, pp. 384–388.

[28] GLOBUS, A., LAWTON, J., AND WIPKE, T. Automatic molecular design using evolutionary techniques. *Nanotechnology 10*, 3 (1999), 290.

[29] GLOVER, F. Tabu search-part i. *ORSA Journal on computing 1*, 3 (1989), 190–206.

[30] GOTTSCHALK, K., GRAHAM, S., KREGER, H., AND SNELL, J. Introduction to web services architecture. *IBM systems Journal 41*, 2 (2002), 170–177.

[31] GUO, G., YU, F., CHEN, Z., AND XIE, D. A four-level matching model for semantic web service selection based on qos ontology. In *Information Science and Engineering (ISISE), 2010 International Symposium on* (2010), IEEE, pp. 630–634.

[32] HART, E., ROSS, P., AND CORNE, D. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines 6*, 2 (2005), 191–220.

[33] HUANG, Z., JIANG, W., HU, S., AND LIU, Z. Effective pruning algorithm for qos-aware service composition. In *Commerce and Enterprise Computing, 2009. CEC'09. IEEE Conference on* (2009), IEEE, pp. 519–522.

[34] 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).

[35] JIANG, W., ZHANG, C., HUANG, Z., CHEN, M., HU, S., AND LIU, Z. Qsynth: A tool for qos-aware automatic service composition. In *Web Services (ICWS), 2010 IEEE International Conference on* (2010), IEEE, pp. 42–49.

[36] JOSHI, A., ROWE, J. E., AND ZARGES, C. Improving the performance of the germinal center artificial immune system using\ epsilon-dominance: A multi-objective knapsack problem case study. In *Evolutionary Computation in Combinatorial Optimization*. Springer, 2015, pp. 114–125.

[37] KANTSCHIK, W., AND BANZHAF, W. Linear-graph gp-a new gp structure. In *Genetic Programming*. Springer, 2002, pp. 83–92.

[38] KARAKOC, E., AND SENKUL, P. Composing semantic web services under constraints. *Expert Systems with Applications 36*, 8 (2009), 11021–11029.

[39] KATTEPUR, A., GEORGANTAS, N., AND ISSARNY, V. Qos composition and analysis in reconfigurable web services choreographies. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 235–242.

[40] KAZHAMIAKIN, 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.

[41] KHAKHKHAR, S., KUMAR, V., AND CHAUDHARY, S. Dynamic service composition. *International Journal of Computer Science and Artificial Intelligence* (2012).

[42] KO, J. M., KIM, C. O., AND KWON, I.-H. Quality-of-service oriented web service composition algorithm and planning architecture. *Journal of Systems and Software 81*, 11 (2008), 2079–2090.

[43] 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.

[44] 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.

[45] 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.

[46] LI, W., BADR, Y., AND BIENNIER, F. Towards a capability model for web service composition. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 609–610.

[47] LI, X., YANG, G., AND HIRASAWA, K. Evolving directed graphs with artificial bee colony algorithm. In *Intelligent Systems Design and Applications (ISDA), 2014 14th International Conference on* (2014), IEEE, pp. 89–94.

[48] LIN, N., KUTER, U., AND SIRIN, E. *Web service composition with user preferences*. Springer, 2008.

[49] 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 2005*. Springer, 2005, pp. 84–89.

[50] LU, J., YU, Y., ROY, D., AND SAHA, D. Web service composition: A reality check. In *Web Information Systems Engineering–WISE 2007*. Springer, 2007, pp. 523–532.

[51] 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.

[52] MARCONI, A., PISTORE, M., AND POCCIANTI, P. Automatedweb service composition at work: the amazon/mps case study. In *Web Services, 2007. ICWS 2007. IEEE International Conference on* (2007), IEEE, pp. 767–774.

[53] MARCONI, A., PISTORE, M., AND TRAVERSO, P. Specifying data-flow requirements for the automated composition of web services. In *Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on* (2006), IEEE, pp. 147–156.

[54] 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.

[55] MAUÉ, P., AND ROMAN, D. The envision environmental portal and services infrastructure. In *Environmental Software Systems. Frameworks of eEnvironment*. Springer, 2011, pp. 280–294.

[56] MENASCE, D. Qos issues in web services. *Internet Computing, IEEE 6*, 6 (2002), 72–75.

[57] MILANOVIC, N., AND MALEK, M. Current solutions for web service composition. *IEEE Internet Computing 8*, 6 (2004), 51–59.

[58] MILLER, J. F., AND THOMSON, P. Cartesian genetic programming. In *Genetic Programming*. Springer, 2000, pp. 121–132.

[59] MOBEDPOUR, D., AND DING, C. User-centered design of a qos-based web service selection system. *Service Oriented Computing and Applications 7*, 2 (2013), 117–127.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] PERREY, R., AND LYCETT, M. Service-oriented architecture. In *Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on* (2003), IEEE, pp. 116–119.

[65] POLI, R. *Parallel distributed genetic programming*. Citeseer, 1996.

[66] 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.

[67] 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).

[68] POP, C. B., VLAD, M., CHIFU, V. R., SALOMIE, I., AND DINSOREANU, M. A tabu search optimization approach for semantic web service composition. In *Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on* (2011), IEEE, pp. 274–277.

[69] 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.

[70] 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.

[71] 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.

[72] SHIN, D.-H., AND LEE, K.-H. An automated composition of information web services based on functional semantics. In *Services, 2007 IEEE Congress on* (2007), IEEE, pp. 300–307.

[73] SKIENA, S. Dijkstra's algorithm. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley* (1990), 225–227.

[74] SOHRABI, S., PROKOSHYNA, 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.

[75] SOYDAN BILGIN, A., AND SINGH, M. P. A daml-based repository for qos-aware semantic web service selection. In *Web Services, 2004. Proceedings. IEEE International Conference on* (2004), IEEE, pp. 368–375.

[76] 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.

[77] 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.

[78] WANG, L., SHEN, J., AND YONG, J. A survey on bio-inspired algorithms for web service composition. In *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on* (2012), IEEE, pp. 569–574.

[79] 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.

[80] WANG, X., VITVAR, T., KERRIGAN, M., AND TOMA, I. A qos-aware selection model for semantic web services. In *Service-Oriented Computing–ICSOC 2006*. Springer, 2006, pp. 390–401.

[81] WANG, X., WANG, Z., AND XU, X. An improved artificial bee colony approach to qos-aware service selection. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 395–402.

[82] 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.

[83] 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 2003*. Springer, 2003, pp. 200–215.

[84] XIA, Y.-M., AND YANG, Y.-B. Web service composition integrating qos optimization and redundancy removal. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 203–210.

[85] XIANG, C., ZHAO, W., TIAN, C., NIE, J., AND ZHANG, J. Qos-aware, optimal and automated service composition with users' constraints. In *e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on* (2011), IEEE, pp. 223–228.

[86] 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.

[87] 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).

[88] 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.

[89] YU, Q., AND BOUGUETTAYA, A. Efficient service skyline computation for composite service selection. *Knowledge and Data Engineering, IEEE Transactions on 25*, 4 (2013), 776–789.

[90] YU, Y., MA, H., AND ZHANG, M. An adaptive genetic programming approach to qos-aware web services composition. In *Evolutionary Computation (CEC), 2013 IEEE Congress on* (2013), IEEE, pp. 1740–1747.

[91] ZHANG, S., DOU, W., AND CHEN, J. Selecting top-k composite web services using preference-aware dominance relationship. In *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 75–82.

[92] ZHANG, W., CHANG, C. K., FENG, T., AND JIANG, H.-Y. Qos-based dynamic web service composition with ant colony optimization. In *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual* (2010), IEEE, pp. 493–502.

[93] 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 *Web Services (ICWS), 2013 IEEE 20th International Conference on* (2013), IEEE, pp. 605–606.