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

Comprehensive Quality-Aware Semantic Web Service Composition

Chen Wang

Supervisors: Dr. Hui Ma, Dr. Aaron Chen

June 16, 2017

Submitted in partial fulfilment of the requirements for
PhD.

Abstract

......

Contents

Figures

Chapter 1

Introduction

1.1 Problem Statement

Service-oriented computing (SOC) is a novel computing paradigm that employs services as fundamental elements to achieve the agile development of cost-efficient and integrable enterprise applications in heterogeneous environments [77, 78]. One of the primary purposes of SOC is to overcome conflicts due to diverse platforms and programming languages to make integrable and seamless communication among those existing or newly built independent services. *Service Oriented Architecture* (SOA) could abstractly realise service-oriented paradigm of computing. This accomplishment has been contributing to reuse of software components, from the concept of functions to units and from units to services during the evolution of development in SOA [10, 74]. One of the most typical implementation of SOA is *web service*, which designated as "modular, self-describing, self-contained applications that are available on the Internet" [21]. Several standards play a significant role in registering, enquiring and grounding web services across the web, such as UDDI [20], WSDL [52] and SOAP [32]. *Web service composition* aims to loosely couple a set of web services to provide a value-added composite service that accommodates complex functional and non-functional requirements of service users.

Two most notable challenges for web service composition are ensuring interoperability of services and achieving Quality of Service (QoS) optimisation [32]. *Interoperability* of web services presents challenge in syntactic and semantic dimensions. The syntactic dimension is covered by the XML-based technologies [109], such as the previously discussed *WSDL* and *SOAP*. In this dimension, most of service compositions are merely based on the matching of input-output parameters. The semantic dimension enables a better collaboration through ontology-based semantics [73], in which many standards have been established in this dimension. E.g., OWL-S [66], Web Service Modeling Ontology (WSMO) [53], SAWSDL [50], Semantic Web Services Ontology (SWSO) [81]. This dimension bring around some other services' resources that could effect the execution of web services and their composition (i.e., precondition and postcondition). On the whole, these two challenges give birth to *Semantic web services composition* and *QoS-aware service composition*. *Semantic web services composition* is distinguished from the tranditional service composition (i.e., only syntactic dimension presented in web services). The resources of semantic web services are described semantically to enable a better interoperability for chaining web services. Another challenge is related to QoS optimisation. E.g., minimum cost and maximum reliability. This problem gives birth to *QoS-aware service composition* that aims to find composition solutions with optimised QoS.

Further more, the environment of service composition is changing in the real world, rather than *static*. E.g, QoS values of services being composed of are fluctuating over time, or service chosen at the planning stage may not avaible to be invoked at the runtime. Most of importance is *static web service composition* supports the environment change badlly because of outdated composition solutions. Therefore, *Dynamic web service composition* become a very demanding research field with a growing interest for providing solutions that adapt to the changing environment. Additionly, in context of semantic web service composition, semantics of web services can make the problem of dynamic web service composition more complicated for the changing ontology.

Different approaches have been proposed to solve those composition problems discussed above and they can be classified into two main categories: *semi-automated web service composition* and *fully automated web service composition*. The first composition problem requires human beings to manually create abstract workflows. Generally, researchers assume the pre-defined abstract workflow is given and provided by the users. The optimisation problem in this approach turns to selecting the concrete services with the best possible quality to each slot of the given workflow. Due to a tremendous growth in industries and enterprise applications, the number of web services has increased dramatically and unprecedentedly. The process of conducting abstract workflows manually is fraught with difficulties. Therefore, fully automation of the composition process was introduced in web service composition for less human intervention, less time consumption, and high productivity. The advantages of fully automated approach is that an abstract workflow is not not provided, but it is established while service are selected.

Generating composition plans automatically in discovering and selecting suitable web services is a NP-hard problem [71], which means the composition solution is not likely to be found with reasonable computation times in a large searching space. *Artificial Intelligence (AI) planning-based approaches*, *Evolutionary Computation (EC) techniques* and hybrid techniques are introduced to handle this problem. AI planning problem is solved in making a plan, from initial states to a set of actions to desired goal states-composite web services, where services are considered as actions triggered by one state (i.e., inputs) and resulted in another state (i.e., outputs). In the second approach, heuristics have to be employed to generate near-optimal solutions, where a variety of EC techniques have been used in this context. E.g., Genetic Algorithms (GA), Genetic Programming (GP) and Particle Swarm Optimisation (PSO). EC-based techniques have been effectively proposed to solve *QoS-aware web service composition* problems with different designed data structures for representation. Recently, different solution representation utilised in different EC-based method have been investigated in QoS-aware web service composition problems, since they could significantly impact on the performance while performing fully automated service composition. In the third approach, a hybrid of AI planning-based approaches and EC-based approaches are proposed to fulfil the correctness in constructing workflows with users' constraints, while the quality of composition solutions are also optimised according to users' requirement.

The overall goal of this thesis is to propose a comprehensive-quality aware automated web service composition. This comprehensive quality aims to jointly optimise semantic matchmaking quality and QoS. Meanwhile, this new approach also tackles serveral service composition problems, such as semantic web service compostion, multi-objective optimisation, dynamic web service composition.

1.2 Motivations

The motivations of automated semantic web service composition lies in the requirements from three key aspects that simultaneously account for. 1. *Quality of service composition*: previous studies have suggested many approaches to optimise QoS, which refers to the non–

functional quality of service composition. However, the importance of the functional quality (i.e, semantic matchmaking quality) are not recognised in these works. 2. *Composition constructs*: 3. *dynamic service composition*: Herein these requirements are explicitly discussed below.

Hybridisation of Quality of Semantic Matchmaking and Quality of Service

Often, many different service compositions can meet a user request but differ significantly in terms of QoS and semantic matchmaking quality. For example, in the classical travel planning context, some component service must be employed to obtain a travel map. Suppose that two services can be considered for this purpose. One service S can provide a street map at a price of 6.72. The other service S' can provide a tourist map at a price of 16.87. Because in our context a tourist map is more desirable than a street map, S' clearly enjoys better semantic matchmaking quality than S but will have negative impact on the QoS of the service composition (i.e., the price is much higher). One can easily imagine that similar challenges frequently occur when looking for service compositions. Hence, a good balance between QoS and semantic matchmaking quality is called for.

Existing works on service composition focus mainly on addressing only one quality aspect discussed above. For the semantic matchmaking quality, it is mainly addressed in the discovery of an atomic service. I.e., one-to-one matching of user requirements to a single service. The service composition [6, 11, 68] utilised semantic descriptions of web services (e.g., description logic) to ensure the interoperability of web services, but the goal of the composition is often to minimise the number of services or the size of a graph representation for a web service composition. These approaches do not guarantee an optimised QoS of service compositions. On the other hand, huge efforts have been devoted to studying QoS-aware web service composition, and some approaches to QoS-aware web service composition do consider the semantic matchmaking while composing solutions, but they do not recognise the importance of semantic matchmaking quality. For these reasons, there is a need to device an automated approach for jointly optimising the two quality aspects.

Multi-Objective Composition Optimisation

Web service compostion problems fall into two groups, depending on an optimisation for a single objective or multiple objectives. In single-objective service compositions, a single composition solution is always returned by the composition task, when the preferences of each quality conponent within the single objective (e.g., a weighted sum of different quality criteria) is known by users. However, users do not always have clear preferences when many quality criteria are presented. Therefore, multi-objective is a natural features of requirements from users to provide a set of trade-off solutions for the conflicting quality criteria. E.g., Premium users do not care cost as much as price-sensitive users do, so they may prefer a composition solution with lowest execution time, rather than one with a relatively lower execution time without exceeding a budget. Therefore, a multi-objective fully automated service composition approach is very demanding.

Existing research on the automated web service compostion mainly contrates on sigle objective problems for QoS-aware web service compositions. I.e., there is only one solution promoted by a unified QoS ranking score to all the users. However, in multi-objective context, service composition problems [60, 95, 106] are only approached by semi-automated methods, where the workflow structure is assumed to be pre-existing. These approaches handle the conflicting QoS attributes independently. From above discussion, these is a lack of fully automated approaches to multi-objective web service composition problems. Moreover, the insufficiency of handling only non-functional attributes (i.e., QoS) has given rise to adding semantic matchmaking quality into consideration.

Automated Web Service Composition Based on Preconditions and Effects

Apart from considering the satisfactory inputs and production of outputs, some conditional constraints also determine the execution of services. These conditional constraints lead to multiple possible paths for execution when services are composed together. E.g., In the scenario of an online book shopping system adapted from [99], services are composed to provide an operation for book shopping. Users expect that purchase outcome (e.g. receipt) is returned If book and customer details (e.g. title, author, customer id) are given. In this case, the users may have specific constraints. 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 instalments. The constraints are on their current account balance.

Most of the approaches to automated web service composition are approached through services represented by only inputs and outputs, which are simply utilised to complete service composition. While the underlying functional knowledge of services (i.e., prerequisites for execution, and result in some changes, often know as precondition and effects) is not included [75]. Despite many promising approaches [12] been explored to achieve compositions that consider precondition and effects using AI planning. Evolutionary Computation (EC) techniques are considered to be more flexible and more efficient, but they have not been widely explored for automated web service composition based on precondition and effects.

Dynamic Semantic Web Service Composition

In a dynamic environment, QoS of the atomic services in service repository is a moving target. Static service composition solution is no longer enough, and requisite measures must be taken if the composition solution may decrease in QoS or not be executable if any service involved goes offline. Therefore, dynamic Web service composition is proposed to monitor and update composition solutions when they are antiquated [55].

The majority of techniques endeavoured to update antiquated or incorrect compositions allow for dynamic adaptation of the solutions based on variability model [2]. However, most of the EC-based approaches to web service composition have been studied in static scenarios, rather than dynamic ones. Although a lack of research in this field, they have been showing its promise for the confidence of its behaviour for handling dynamic web service composition: a proper amount of population stored could be used for retrieving an alternative composition solution in the case of failure. Also, The stored population could be further evolved while taking changes of QoS into account. This self-heal process supports the adaption of a dynamic environment. Presented those benefits, it is advisable to study the effectiveness of EC approaches in a dynamic composition context.

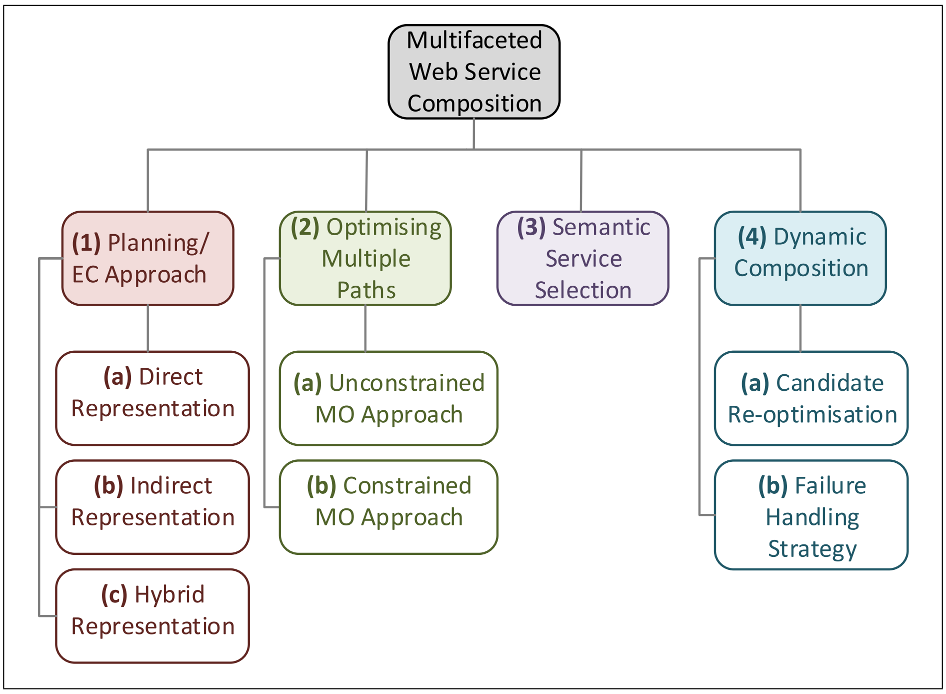


Figure 1.1: Research objectives and sub-objectives.

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. multiple constructs

2. multiple constructs. quality for precondition and effects

3. 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 technique 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 [40, 19]. 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 [23].

(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 3a and 3b, 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.

4. 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.

5. Develop an EC technique for performing semantic Web service selection in the context of a planning-based composition technique. As discussed in Objective 3, 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.

6. 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 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 3.3, 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 [71]. 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.

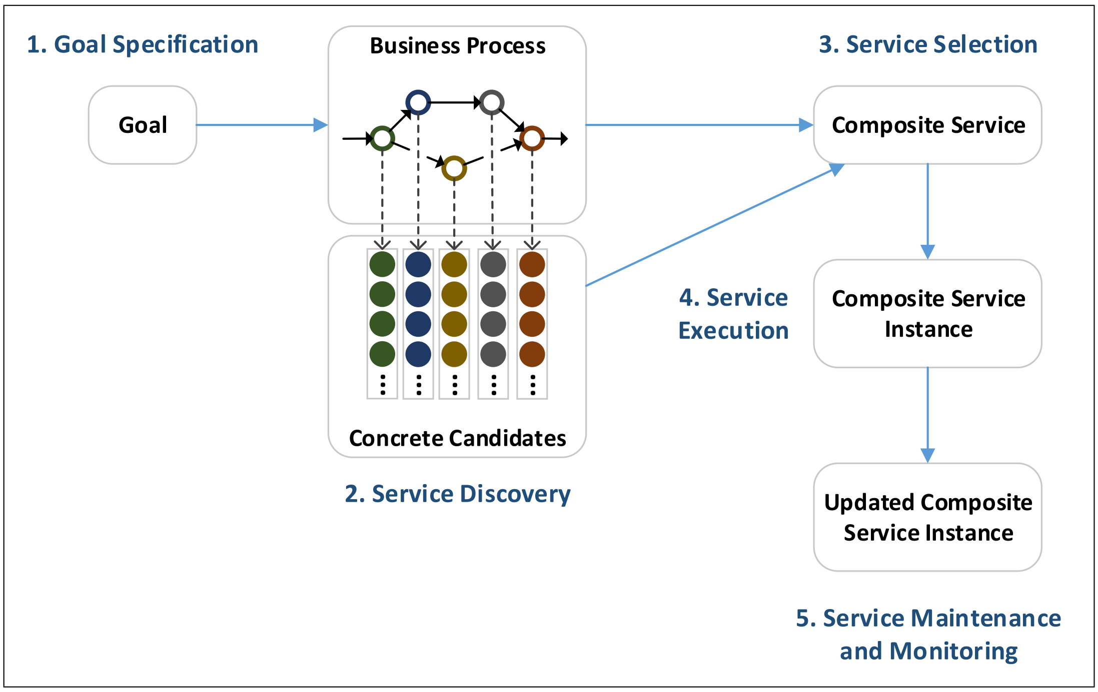


Figure 2.1: Typical steps in a workflow-based automated Web service composition solution [71].

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 [86]. 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 [23], 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 [23]. The majority of works in this area use maximising and minimising functions for the different

QoS attributes, meaning that they attempt to obtain solutions with the best possible qualities without any thresholds [101, 79, 107, 36, 112, 98, 71, 47, 56]. 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) [107, 106, 16, 7].

The second group comprises the creation of service compositions that have multiple execution branches, indicating user preferences at runtime [99, 13, 46, 65, 8]. 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 [61] 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 [64]. 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 [70] 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 [98]. 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 [98]. 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 [112]. 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,

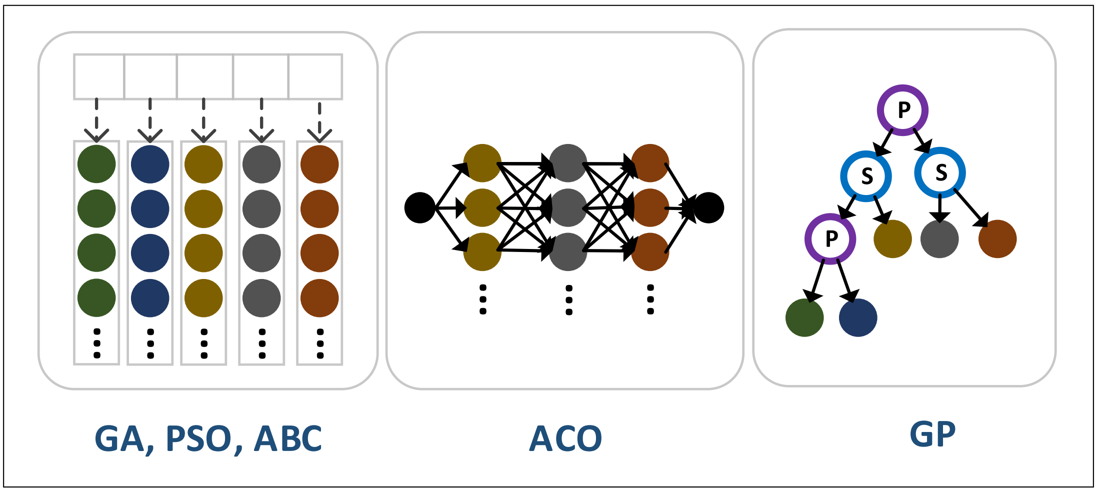


Figure 2.2: The different problem representations employed by biologically-inspired Web service composition approaches.

and also a heuristic function for that particular graph. The pheromones left by the ants are updated after all ants have toured through the graph once, with paths of higher fitness resulting in a larger pheromone increment for the 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 [112] and [101] 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 interesting 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, 87], 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 [19], 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 [34] 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 [69] 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 [63] 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 [58] 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 [82] 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 [45] 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 [37, 14, 72] 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** [38] 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 [79]. 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 [79, 5], which restricts the problem model to address semi-automated compositions only.

Integer Linear Programming (ILP) has also been applied to composing Web services [107]. 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 [56] 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 [33, 47]. 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 [33], involves constructing a mapping between algebraic expression and BPEL, to allow for an automated translation between these two representations. The work in [47] 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) [42] 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 [23] is a classic example of this SAW technique:

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

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 [60, 112, 108, 106, 105, 16]. 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 [60], however other EC algorithms have also been considered. The work in [112], 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 [106], 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 [111], 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 [111]. 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 [28] 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 [27], and thus alternative techniques must be investigated to handle such scenarios. These are known as **many-objective techniques**. The work discussed in [27] 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

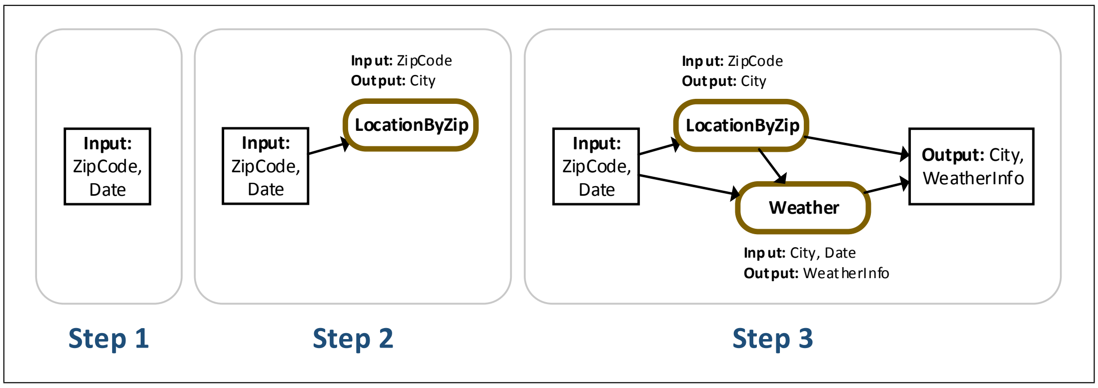


Figure 2.3: Basic example of Web service composition using the Graphplan algorithm.

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, 44].

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 [31, 96, 103, 99] have been based on a fast planning algorithm named Graphplan [9]. 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 [15], 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 [92] 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 [29] 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 [41]. 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 [43], 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 [8]. The work in [93] 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 [48]. 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 [65]. 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 [18, 85, 104, 17]. 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 conditional 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 [83], 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 [84] 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 [20], 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) [67] 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) [50], 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 [94, 100, 36, 59, 46, 13, 88, 57, 113]. The work in [12], 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 [91]. 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 [39] proposes a semantic Web service selection method that performs matching 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 [35]. 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, 7, 102]. The work in [49] 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 [97]. 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 [30] 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 [80]. 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

This chapter presents the highlights of the initial work conducted on the creation of a Web service composition approach that combines a planning algorithm, for candidate creation and modification, with EC techniques, for optimising candidates according to their overall Quality of Service. In particular, two types of direct solution representations are explored in alignment with Objective 1: a tree-based representation, which is compatible with the existing Genetic Programming algorithm but may lead to service duplication problems, and a graph-based representation, which prevents duplication issues but requires a more complex evolutionary algorithm. These two approaches are discussed and compared in the following sections.

3.1 Motivation and Problem Description

3.1.1 Motivation

Our goal is to develop an automated approach for generating good service compositions. Often, many different service compositions can meet a user request but differ significantly in terms of QoS and semantic matchmaking quality. A motivating example of classical travel planning context is shown in Fig 3.1. The figure shows a DAG that represents a solution with four web services involved for a request R as a composition tast. The inputs of the task are $\{TravelDepartureDate, HomeCity, ConferenceCity, TravelReturnDate\}$ and outputs are $\{BusTicket, FlightTicket, TouristMap, HotelReservation\}$. This is a classic example of travel planning problem for people who seek for booking services of airplanes, buses and hotels reservation, and also generating tourist maps for the conference city.

Although obtaining a correct combination of web service is essential, there are many ways to connect multiple web services chosen from a service repository. These web services provide different QoS in terms of availability, price, reliability and response time. Moreover, valid semantic matches between two connected web services could also have different matchmaking quality. For example, *GenerateMap* service produce *StreetMap* which is considered to be a valid semantic match to *TouristMap*, while *BusService* produce *BusTicket* which is considered to be a better valid semantic match to *BusTicket*. The descriptions of these matched resources are captured in a ontology depicted in Fig .3.2, where all the inputs and the outputs related to the web services are described for the travel planning problem discuss here. For example, the concept *Map* and its sub-concept *urbanMap* are assigned with a *touristMap* instance and *streetMap* instance respectively in Fig .3.2. The valid semantic match is considered that an instance of *urbanMap* can be an instance of *Map*, but not vice versa. In the example of the travel planning problem, some component service must be employed to obtain a travel map. one service *GenerateMap* can provide a street

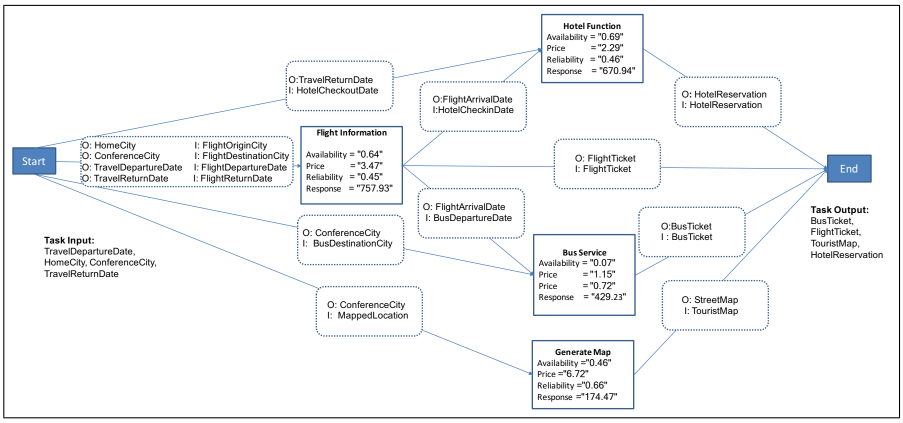


Figure 3.1: An example of a web service composition

map at a price of 6.72. The other service *GenerateTouristMap* can provide a tourist map at a price of 16.87. Because in our context a tourist map is more desirable than a street map, *GenerateTouristMap* clearly enjoys better semantic matchmaking quality than *GenerateMap* but will have negative impact on the QoS of the service composition (i.e., the price is much higher). One can easily imagine that similar challenges frequently occur when looking for service compositions. Hence, a good balance between QoS and semantic matchmaking quality is called for. We therefore propose a *comprehensive quality model* in considering semantic matchmaking quality and QoS simultaneously.

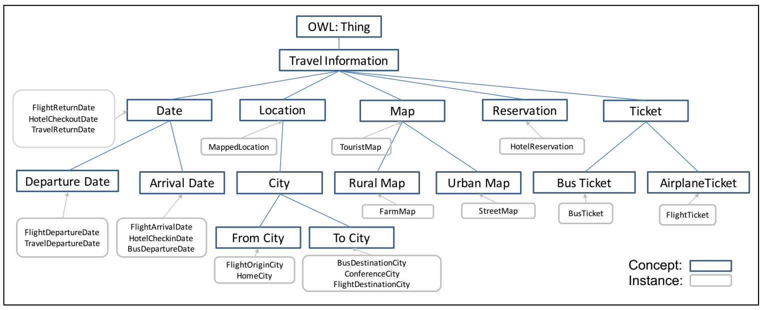


Figure 3.2: An ontology of a travel planning domain

3.2 Problem Description and Comprehensive Quality Model

We consider a *semantic web service* (*service*, for short) as a tuple $S = (I_S, O_S, QoS_S)$ where I_S is the set of service inputs that are consumed by S, O_S the set of service outputs that are produced by S, and $QoS_S = \{t_S, c_S, r_S, a_S\}$ the set of non-functional attributes of S. The inputs in I_S and outputs in O_S are parameters that are related to concepts in an ontology $\mathcal{O}$. The attributes t_S, c_S, r_S, a_S refer to the response time, cost, reliability, and availability of service S, respectively. These four QoS attributes are most commonly used [110].

A *service repository* $\mathcal{SR}$ is a finite collection of services with a common ontology $\mathcal{O}$. A *service request* (also called *composition task*) over $\mathcal{SR}$ is a tuple $T = (I_T, O_T)$ where I_T is the set of task inputs, and O_T the set of task outputs. The inputs in I_T and outputs in O_T are parameters that are related to concepts in the ontology $\mathcal{O}$.

A service composition is commonly represented as a *directed acyclic graph* (DAG). It nodes correspond to the services in the composition. Two services S and S' are connected by an edge e if some output of S serves as input for S'. Apparently, such outputs and inputs must semantically match to ensure the correct execution of the service composition. The mechanism to compose services relies on the semantic descriptions of inputs and outputs, which enables inputs of services to be matched by outputs of other services. The following *matchmaking types* are often used to describe the level of a match [76]: For concepts a, b in $\mathcal{O}$ the *matchmaking* returns *exact* if a and b are equivalent ($a \equiv b$), *plugin* if a is a sub-concept of b ($a \sqsubseteq b$), *subsume* if a is a super-concept of b ($a \sqsupseteq b$), and *fail* if none of previous matchmaking types is returned. In this paper we are only interested in robust compositions where only *exact* and *plugin* matches are considered, see [54].

As argued in [54] *plugin* matches are less preferable than *exact* matches due to the overheads associated with data processing. We suggest to consider the semantic similarity of concepts when comparing different *plugin* matches. For concepts a, b in $\mathcal{O}$ the *semantic similarity* $sim(a, b)$ is calculated based on the edge counting method defined in Eq. (3.1) from [89], where N_a, N_b and N_c measure the distances from concept a, concept b, and a closest common ancestor c of a and b to the top concept of the ontology $\mathcal{O}$, respectively.

$$sim(a, b) = \frac{2N_c \cdot e^{-\lambda L/D}}{N_a + N_b} \tag{3.1}$$

For our purposes, λ can be set to 0 as we do not measure the similarities of neighbourhood concepts, which is not the matching type considered in this paper.

Given a service request $T = (I_T, O_T)$, we represent a service composition solution for T with services $S_1, \ldots, S_n$ by a weighted DAG, $WG = (V, E)$ with node set $V = \{Start, S_1, S_2, \ldots, S_n, End\}$ and edge set $E = \{e_1, e_2, ...e_m\}$. $Start$ and End are two special services defined as $Start = (\emptyset, I_T, \emptyset)$ and $End = (O_T, \emptyset, \emptyset)$ that account for the input and output requirements given by the request. Each edge e from a service S to a service S' means that service S produces an output $a \in O_S$ that is matched (*exact* or *plugin*) to an input $b \in I_{S'}$ to be consumed by service S' in the composition. Based on the matchmaking type the *semantic matchmaking quality* of edge e can be defined as follows:

$$type_e = \begin{cases} 1 & \text{if } a \equiv b \text{ (\textit{exact} match)}, \\ p & \text{if } a \sqsubseteq b \text{ (\textit{plugin} match)} \end{cases} \tag{3.2}$$

$$sim_e = sim(a, b) = \frac{2N_c}{N_a + N_b} \tag{3.3}$$

with a suitable parameter $p, 0 < p < 1$ to chosen, for discussion see Section 3.3.1. However, if more than one pair of matched output and input exist from services S and S' respectively, $type_e$ and sim_e will take on their average values.

The *semantic matchmaking quality* of the service composition can be obtained by aggregating overall edges in E as follows:

$$MT = \prod_{j=1}^{m} type_{e_j} \tag{3.4}$$

$$SIM = \frac{1}{m} \sum_{j=1}^{m} sim_{e_j} \tag{3.5}$$

The *QoS* of the service composition can be obtained by aggregating the QoS values of the participating services. For a service composition with services $S_1, S_2, ...S_n$ we obtain the reliability $R = \prod_{k=1}^{n} r_{S_k}$, the availability $A = \prod_{k=1}^{n} a_{S_k}$, the cost $C = \sum_{k=1}^{n} c_{S_k}$, and the response time T is the time of most time-consumption path in the composition, i.e.,

$$T = MAX\{\sum_{k=1}^{\ell_j} t_{S_k} | \; j \in \{1, \dots, m\} \text{ and } \ell_j \text{ is the length of a path } P_j\}.$$

When multiple quality criteria are involved into decision making, then the overall fitness of a solution can be defined as a weighted sum of the individual criteria:

$$Fitness = w_1\hat{MT} + w_2\hat{SIM} + w_3\hat{A} + w_4\hat{R} + w_5(1-\hat{T}) + w_6(1-\hat{C}) \tag{3.6}$$

with $\sum_{k=1}^{6} w_k = 1$. We call this objective function the *comprehensive quality model* for service composition. The weights can be adjusted according to users' preferences. Herein, the individual criteria are normalised to a range between 0 to 1, where 1 means the best value and 0 means the worst. For this purpose, we normalise MT, SIM, A, R, T, and C so that the function value falls within the range from 0 to 1 using Eq. (3.7). To simplify the presentation we also use the notation $(Q_1, Q_2, Q_3, Q_4, Q_5, Q_6) = (MT, SIM, A, R, T, C)$. MT and SIM have minimum value 0 and maximum value 1. The minimum and maximum value of A, R, T, and C are calculated across all task-related candidates in the service repository $\mathcal{SR}$ using greedy search, for details see Section 3.3.1.

$$\hat{Q}_k = \begin{cases} \frac{Q_k - Q_{k,min}}{Q_{k,max} - Q_{k,min}} & \text{if } k = 1, \dots, 4 \text{ and } Q_{k,max} - Q_{k,min} \neq 0, \\ \frac{Q_{k,max} - Q_k}{Q_{k,max} - Q_{k,min}} & \text{if } k = 5, 6 \text{ and } Q_{k,max} - Q_{k,min} \neq 0, \\ 1 & \text{otherwise.} \end{cases} \tag{3.7}$$

To solve the composition task best possible the goal is to find the maximum value of the objective function in Eq. (3.6).

3.3 PSO-based Approach

3.3.1 An Overview of our PSO-based Approach

As PSO has shown promise in solving combinatorial optimisation problems, we propose a PSO-based approach to comprehensive quality-aware automated semantic web service composition. Fig. 3.3 shows an overview of our approach consisting of four steps:

Step 1: The composition process is triggered by a composition task, which is clearly defined in Section **??**.

Step 2: The composition task is used to discover all task-related service candidates using a greedy search algorithm adopted from [62], which contributes to a shrunken service repository. This greedy search algorithm keeps adding outputs of the invoked services as available outputs (initialised with I_T) , and these available outputs are used to discover task-related services from a service repository and updated with the outputs of these discovered services. This operation is repeated until no service is satisfied by the available outputs. During the greedy search, an ontology-based cache (*cache*) is initialised, which stores the concept similarities of matched inputs and outputs of task-related candidates. This *cache* is also used to discover services by checking whether *null* is returned by given two output-related and input-related concepts.

Step 3 and Step 4: These two steps follow the standard PSO steps [90] except for some differences in particles mapping and decoding processes. In particular, these two differences are related to sorting a created service queue using service-to-index mapping for a particle' position vectors and evaluating the fitness of a particle after decoding this service queue into a WG respectively. Those differences are further addressed in Algorithms 1 and 2 in Section 3.3.2.

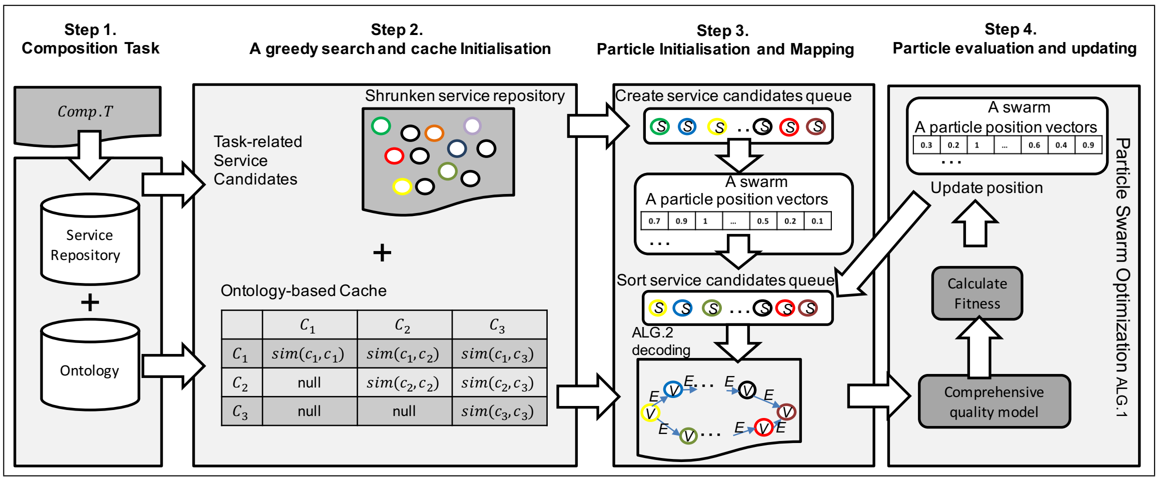


Figure 3.3: An overview of our PSO-based approach to comprehensive quality-aware automated semantic web service composition.

3.3.2 The Algorithms for our PSO-based Approach

The overall algorithm investigated here is made up of a PSO-based web service composition technique (Algorithm 1) and a *WG* creating technique from a service queue (Algorithm 2). In Algorithm 1, the steps 4, 5, 6 and 7 are different from those of standard PSO: In step 4, the size of task-related service candidates generated by a greedy search determines the size of each particle's position. Each service candidate in a created service candidates queue is mapped to an index of a particles position vectors, where each vector has a weight value between 0.0 and 1.0. In step 5, service candidates in the queue are sorted according to their corresponding weight values in descending order. In step 6, this sorted queue is used as one of the inputs of the forward decoding Algorithm 2 to create a *WG*. In step 7, the fitness value of the created *WG* is the fitness value of the particle calculated by the comprehensive model discussed in Section **??**.

ALGORITHM 1. Steps of PSO-based service composition technique [26].

1: Randomly initialise each particle in the swarm;
2: **while** *max. iterations not met* **do**
3:     **foreach** *particle in the swarm* **do**
4:         Create a service candidates queue and map service candidates to a particle's position vectors;
5:         Sort the service queue by position vectors' weights;
6:         Use Algorithm 2 to create a *WG* from the service queue ;
7:         Calculate the *WG* fitness value;
8:         **if** *fitness value better than pBest* **then**
9:             Assign current fitness as new *pBest*;
10:        **else**
11:             Keep previous *pBest*;
12:     Assign best particle's *pBest* value to *gBest*, if better than *gBest*;
13:     Calculate the velocity of each particle;
14:     Update the position of each particle;

Algorithm 2 is a forward graph building algorithm extended from [9]. This algorithm takes one input, a sorted service queue from step 5 of Algorithm 1. Note that different service queues may lead to different *WGs*. In addition. I_T, O_T and *cache* are also taken as the inputs. Firstly, *Start* and *End* are added to V of WG as an initialisation, and *OutputSet* is also created with I_T. The following steps are repeated until O_T can be satisfied by *Outputset* or the service queue is *null*. If all the inputs I_S of the first popped S from *queue* can be satisfied by provided outputs from *OutputSet*, this S is added to V and its outputs are added to *OutputSet*, and S is removed from *queue*. Otherwise, the second popped S from *queue* is considered for these operations. Meanwhile, e is created with $type_e$ and sim_e if S is added, and calculated using information provided from *cache*. This forward graph building technique could lead to more services and edges connected to the WG, these redundancies should be removed before WG is returned.

ALGORITHM 2. Create a WG from a sorted service queue.

Input : I_T, O_T, *queue*, *cache*
Output: WG

$WG = (V, E)$;
$V \leftarrow \{Start, End\}$;
$OutputSet \leftarrow \{I_T\}$;
while O_T *not satisfied by* $OutputSet$ **do**
  foreach S *in queue* **do**
    if I_S *satisfied by* $OutputSet$ **then**
      insert S into V ;
      adjoin O_S to $OutputSet$;
      queue.remove S;
      $e \leftarrow$ calculate $type_e$, sim_e using *cache*;
      insert e into E;
remove *dangling nodes* and *edges* from WG;
return WG;

3.3.3 Ontology-based Cached Optimisation

In GP, the tree structure representation, which is converted from weighted graphs with edges as the quality of semantic matchmaking and node with QoS. The bottlenecks of our whole approach lie in building weighted graph for initialisation and mutation. In particularly, it refers the size of service repository and cost of semantic matchmaking quality calculation. The size of service of repository could be shrunken into the task-related web services using greedy search. Later on, the cost of semantic quality calculation could be pre-calculated for parameter-related concepts from the task-related webs services. The key idea of the index is to create a map using a pair of keys, output-related concepts and potentially matched input-related concepts with considering different levels of match types, and the map values store mt_p and mt_s. This optimised cache also contributes to less and constant time for weighted graph building for initialisation and mutation.

3.3.4 Genetic Programming Overview

GP [51] is an automated technique for producing and searching computer programs in a solution space, which is considered as a particular application of Genetic Algorithm with a set of different encoded genes. In GP, each individual as a chromosome, which is commonly represented as a tree structure, refers to a candidate solution in a population. The tree structure has a terminal set and a function set, where variables, constants and functions are consisted of respectively. Also, the tree structure is considered be efficiently traversed and evaluated.

Nature evolution and selection of individual in a population are automated simulated in GP [51]. A fitness function is utilised to evaluate the degree of how good (or bad) of each individual after it is evolved. Once all individuals within one generation are all evaluated, three genetic operators consisting of reproduction, crossover, and mutation are involved in to generate next generation. Reproduction operator retains the elite individual without any changes. Crossover operator replaces one node of one individual with another node of another individual. Mutation operator replaces a randomly selected node in an individual. The whole evaluation process will continue until an optimised solution found or a pre-defined number of generation reached. Therefore, We identify the set of terminals, the set of functions, the fitness function and other relevant parameters to perform a GP-based algorithm.

This chapter presents the highlights of the initial work conducted on the creation of a Web service composition approach that combines a planning algorithm, for candidate creation and modification, with EC techniques, for optimising candidates according to their overall Quality of Service. In particular, two types of direct solution representations are explored, in alignment with Objective 3: a *tree-based representation*, which is compatible with the existing Genetic Programming algorithm but may lead to service duplication problems, and a *graph-based representation*, which prevents duplication issues but requires a more complex evolutionary algorithm. These two approaches are discussed and compared in the following sections.

3.3.5 Ontology-based Index Cached Optimisation

our PSO-based approach demands decoding processes from optimised queues to weighted DAGs, a bottleneck of efficiently constructing weighted DAGs lie in building required information of edges, which is related to the cost of $fullmatch$ $(a \Rightarrow b)$ identifications and $sim(a, b)$ calculations between input-related and output-related concepts of service candidates. To efficiently construct weighted DAGs, we calculate $sim(a, b)$ of all $fullmatch(a \Rightarrow b)$ interleaving with the greedy search discussed in Sect 3.3.1. This information is stored in a table structure with special X value if no $fullmatch(a \Rightarrow b)$ is satisfied. Therefore, this optimised cache contributes to less and constant time for weighted DAG building through the evolutionary process.

3.4 Experiment Design

In this section, we employ a quantitative evaluation approach with a benchmark dataset used in [62, 25], which is an augmented version of Web Service Challenge 2009 (WSC09) including QoS attributes. This benchmark dataset contains five tasks with variable numbers of services, and ontologies, which is considered as a challenge dataset for measuring the scalability of our quality evaluation model. Table 3.1 presents the features of the WSC09 dataset. The number of concepts, individuals in the ontology and services in each data set is shown in the second, third, fourth column respectively. Also, we extend all the datasets with QoS attributes from service providers to enable our evaluation. Two objectives of this evaluation are to: (1) evaluate the effectiveness of our PSO-based approach, see comparison test in Section 3.5.1. (2) evaluate the effectiveness of our proposed comprehensive quality model to

Table 3.1: Features of the WSC09 datasets

Dataset	No.Concept	No.Individual	No.Service
WSC09 01	1578	3102	572
WSC09 02	12388	24815	4129
WSC09 03	18573	37316	8138
WSC09 04	18673	37324	8301
WSC09 05	31044	62132	15211

Table 3.2: Mean fitness values for comparing GP-based approach

WSC09	Original $\mathcal{SR}$	Shrunken $\mathcal{SR}$	PSO-based approach	GP-based approach
Task 1	572	80	$0.5592 \pm 0.0128 \uparrow$	0.5207 ± 0.0208
Task 2	4129	140	$0.4701 \pm 0.0011 \uparrow$	0.4597 ± 0.0029
Task 3	8138	153	0.5504 ± 0.0128	$0.5679 \pm 0.0234 \uparrow$
Task 4	8301	330	$0.4690 \pm 0.0017 \uparrow$	0.4317 ± 0.0097
Task 5	15211	237	$0.4694 \pm 0.0008 \uparrow$	0.2452 ± 0.0369

achieve a desirable balance on semantic matchmaking quality and QoS, see comparison test in Section 3.5.2.

The parameters for the PSO are chosen from the settings from [90], In particular, PSO population size is 30 with 100 generations. We run 30 times independently for each dataset. We configure the weights of fitness function to properly balance semantic matchmaking quality and QoS. Therefore, w_1 and w_2 are set equally to 0.25, and w_3, w_4, w_5, w_6 are all set to 0.125. The p of $type_e$ is set to 0.75 ($plugin$ match) according to [54]. In general, weight settings and parameter p are decided by users' preferences.

3.5 Experiment Results

3.5.1 Comparison Test for GP-based vs. PSO-based approach

To evaluate the effectiveness of our proposed PSO-based approach, we compare our PSO-based method with one recent GP-based approach [62] using our proposed comprehensive quality model. We extend this GP-based approach by measuring the semantic matchmaking quality between parent nodes and children nodes. To make a fair comparison, we use the same number of evaluations (3000 times) for these two approach. We set the parameters of that GP-based approach as 30 individuals and 100 generations, which is considered to be proper settings referring to [24].

The first column of Table 3.2 shows five tasks from WSC09. The second and third column of Table 3.2 show the original service repository size and the shrunk service repository size after the greedy search respectively regarding the five tasks. This greedy search helps reducing the original repository size significantly, which contributes to a reduced searching space. The fourth and fifth column of Table 3.2 show the mean fitness values of 30 independent runs accomplished by two methods. We employ independent-samples T tests to test the significant differences in mean fitness value. The results show that the PSO-based approach outperforms the existing GP-based approach in most cases except Task 3. Note that all p-values are consistently smaller than 0.01. Using our PSO-based approach, small changes to sorted queues (particles in PSO) could lead to big changes to the composition solutions. This enables the PSO-based approach to escape from local optima more easily than the GP-based approach.

Table 3.3: Mean values of SM, QoS and sum of SM and QoS for QoS model and comprehensive quality model using PSO-based approach

WSC09		QoS Model	Comprehensive Quality Model
Task1	SM	0.5373 ± 0.0267	0.5580 ± 0.0094 ↑
	QoS	0.5574 ± 0.0156	0.5604 ± 0.0164
	$SM + QoS$	1.0947 ± 0.0423	1.1184 ± 0.0258 ↑
Task2	SM	0.4549 ± 0.0033	0.4630 ± 0.0042 ↑
	QoS	0.4800 ± 0.0012 ↑	0.4772 ± 0.0025
	$SM + QoS$	0.9349 ± 0.0045	0.9402 ± 0.0067 ↑
Task3	SM	0.5538 ± 0.0082	0.6093 ± 0.0054 ↑
	QoS	0.4940 ± 0.0013 ↑	0.4913 ± 0.0009
	$SM + QoS$	1.0478 ± 0.0095	1.1006 ± 0.0063 ↑
Task4	SM	0.4398 ± 0.0037	0.4604 ± 0.0000 ↑
	QoS	0.4845 ± 0.0010 ↑	0.4734 ± 0.0044
	$SM + QoS$	0.9243 ± 0.0047	0.9338 ± 0.0044 ↑
Task5	SM	0.4580 ± 0.0065	0.4639 ± 0.0013 ↑
	QoS	0.4764 ± 0.0005 ↑	0.4750 ± 0.0007
	$SM + QoS$	0.9344 ± 0.0070	0.9389 ± 0.0020

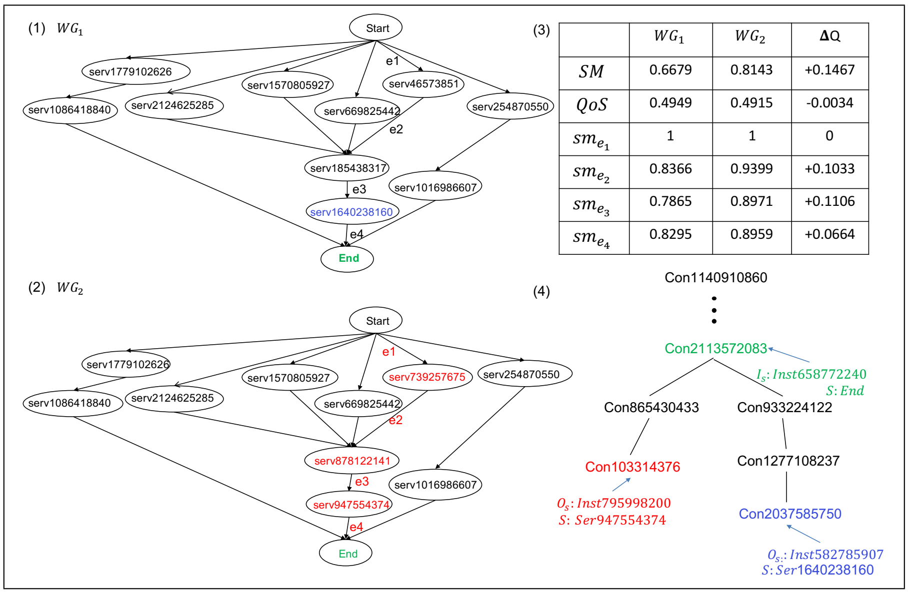
3.5.2 Comparison Test for Comprehensive Quality Model vs. QoS Model

Recently, a QoS Model, $Fitness = w_1\hat{A} + w_2\hat{R} + w_3(1-\hat{T}) + w_4(1-\hat{C})$, where $\sum_{i=1}^{4} w_i = 1$, is widely used for QoS-aware web service composition [62, 26, 22]. To show the effectiveness of our proposed comprehensive quality model, we compare the best solutions found by this QoS model and our comprehensive model using our PSO-based approach. We record and compare the mean values of both SM ($SM = 0.5\hat{MT} + 0.5\hat{SIM}$) and QoS($QoS = 0.25\hat{A} + 0.25\hat{R} + 0.25(1-\hat{T}) + 0.25(1-\hat{C})$) of best solutions over 30 independent runs. To make the comparison informative, all these recorded values have been normalised from 0 to 1, and compared using independent-samples t tests, see Table 3.3.

We observe an interesting pattern from Table 3.3. The mean values of QoS using QoS model are significantly higher than those using comprehensive quality model for Tasks 2, 3, 4 and 5. However, the mean value of SM using the comprehensive quality model are significantly higher than those using the QoS model, while a slight trade-off in QoS are observed in all tasks. In addition, our comprehensive model achieves a consistently higher comprehensive quality in terms of a combination of SM and QoS, which is significantly better in Tasks 1, 2, 3 and 4.

3.5.3 Further Discussion

To analyse the effectiveness of achieving a good comprehensive quality at the expense of slightly reduced QoS, we demonstrate two best solutions produced using Task 3 as an example. Fig. 3.4 (1) and (2) show two weighted DAGs WG_1 and WG_2 that have been obtained as the best service compositions solutions based on the QoS model and on the comprehensive quality model, respectively. Both WGs have exactly the same service workflow structure, but some service vertices and edges denoted in red are different. To better understand these differences, we list the overall semantic matchmaking quality SM, overall QoS and semantic matchmaking quality associated to these different edges in WG_1 and WG_2. (Note: $sm_{e_n} = 0.5type_{e_n} + 0.5sim_{e_n}$), where ΔQ reveals the gain (positive ΔQ) or a loss (negative ΔQ) of the listed qualities for our comprehensive quality model. Therefore, we achieve a comprehensive quality gain of 0.1433, which is the result of a gain in semantic matchmaking quality (+0.1467) and a loss in QoS (-0.0034). To understand the improvement of semantic matchmaking quality from these numbers, we pick up e_4 that is associated with the smallest ΔQ. The e_4 of WG_1 and WG_2 has two different source nodes, $Ser1640238160$ and $Ser947554374$, and two the same end nodes. $Ser1640238160$ and $Ser947554374$ are services with output parameters $Inst582785907$ and $Inst795998200$ corresponds to two concepts $Con2037585750$ and $Con103314376$ respectively in the given ontology shown in Fig. 3.4 (4). As $Inst658772240$ is a required parameter of End, and related to concept $Con2113572083$, $Inst795998200$ is closer to the required output $Inst658772240$ than $Inst582785907$. Therefore, $Ser947554374$ is selected with a better semantic matchmaking quality compared to $Ser1640238160$.



	WG_1	WG_2	ΔQ
SM	0.6679	0.8143	+0.1467
QoS	0.4949	0.4915	-0.0034
sm_{e_1}	1	1	0
sm_{e_2}	0.8366	0.9399	+0.1033
sm_{e_3}	0.7865	0.8971	+0.1106
sm_{e_4}	0.8295	0.8959	+0.0664

Figure 3.4: An example for the comparison of the best solutions obtained based on the QoS model and on the comprehensive quality model for Task 3.

The modified main algorithm $toTree()$ in Algorithm 3 consisting of two stages: cN is located at $Start$ and cN isn't located at $start$. In the first stage, two cases are considered according to the size of cN's outgoing edges. If the size equals 1 (line 3-4), a sequence node is created with a specific $Start$ service node as a right child, and a returned node of $getNextNode()$ (line 26) as a left child. Otherwise(line 5-6), a same sequence node is created with a different left child, a returned node of $parallelNode()$ (line 38). In the second stage, there are two cases (line 13-18 and line 19-24) are involved in according to the size of cN's outgoing edges without considering those target End. Each case are considered under two conditions (line 14 and 20) that there exists one outgoing edge of cN targets End or not. Therefore, same sequence nodes discussed in the first stage are created with different service node cN as a right child, rather than $Start$. If the opposite condition (line 16 and 22) holds, those sequence nodes in the first condition are wrapped as right child of a $parallelNode$, whose left child is a sequence node with a cN service node as its left child and a End service node as right child. We also present the functions for creating sequence node: $seqNode()$ and creating parallel nodes: $parallelNode()$ in Algorithm 3.

ALGORITHM 3. Converting a weighted graph into tree representation.

```
Procedure toTree()
    Input  : current node cN
    Output: tree T
    if cN == Start then
        if |cN.outGoingEdge| = 1 then
            T ← seqNode(servNode("Start"), getNextNode(cN.next));
        else
            T ← seqNode(servNode("Start"), parallelNode(cN));
    else
        foreach oEdge in cN.outGoingEdges do
            if oEdge.target == endNode then
                L ← seqNode(servNode(cN), servNode("O"));
                remove(oEdge);
                break;
        if |cN.outGoingEdge| = 1 then
            if remove(oEdge) == null then
                T ← seqNode(servNode(cN), getNextNode(cN.Next));
            else
                R ← seqNode(servNode(cN), getNextNode(cN.Next));
                T ← parallelNode(L, R);
        if |cN.outGoingEdge| > 1 then
            if remove(oEdge) == null then
                T ← seqNode(servNode(cN), parallelNode(cN));
            else
                L ← seqNode(servNode(cN), parallelNode(cN));
                T ← parallelNode(L, R);
    return T;
Procedure getNextNode()
    Input  : cN
    Output: tree T
    if cN is endNode then
        return seqNode(servNode(cN), servNode("End"));
    else
        toTree();
Procedure seqNode()
    Input  : lChild, rChild
    Output: sequenceNode N
    N ← new seqNode();
    N.child[0] = lChild;
    lChild.parent = N;
    N.child[1] = rChild;
    rChild.parent = N;
    return N;
Procedure parallelNode()
    Input  : cn, cn.outGoingEdges
    Output: parallelNode N
    N ← new ParallelNode();
    N.children[ ] ← new ArrayList[cn.outGoingEdge.size];
    foreach c in N.children[ ] do
        c ← getNextNode(c.Next);
    return N;
```

Chapter 4

Proposed Contributions and Project Plan

4.1 Proposed Contributions

This thesis will contribute to the field of Evolutionary Computation by proposing novel adaptations to candidate representations in an evolutionary algorithms, and to the field of Web service composition by considering multiple composition facets simultaneously. The proposed contributions of this project are listed below:

1. Present a new fully automated Web service composition approach that relies on a combination of planning and EC techniques. This is expected to handle composition tasks that require conditional constraints and quality optimisation simultaneously, a problem that has not yet been investigated by researchers in the area.

2. Extend a many-objective Web service composition approach to handle solutions with conditional constraints, particularly in the case of SLA-aware Web service composition. This extension is expected to enable the optimisation of distinct execution paths independently, meaning that SLA constraints can be verified for each possible execution path. This extension will contribute to an underexplored area in the field of Web service composition.

3. Present an EC-based approach for the semantic selection of Web services in fully automated composition scenarios, as opposed to relying on the typical semi-automated usage where an abstract workflow has already been provided. This novel approach is expected to improve the selection performance of fully automated composition techniques, which currently rely on exhaustive similarity calculations.

4. Introduce the use of Evolutionary Computation techniques in dynamic Web service composition scenarios, leveraging the population diversity intrinsic to these approaches. As the composition environment changes and solutions are re-optimised and adjusted, the use of EC techniques is expected to yield solutions with better quality levels than those produced by approaches that do not perform re-optimisation.

4.2 Overview of Project Plan

Six overall phases have been defined in the initial research plan for this PhD project, as shown in Table 4.1. The first phase, which comprises reviewing the relevant literature, investigating an initial Web service composition approach, and producing the proposal, has

Table 4.1: Overall phases of project plan.

Phase	Description	Duration (months)
1	Reviewing literature, developing initial planning/EC composition approach, and writing the proposal	10 (Complete)
2	Developing direct, indirect, and hybrid planning EC representations	5 (In progress)
3	Investigating many-objective optimisation techniques for the planning/EC approach	6
4	Developing a semantic selection method suitable to planning-based composition approaches	4
5	Adapting the composition approach created so far to function in a dynamic environment	5
6	Writing the thesis	6

Table 4.2: Project timeline for the remaining 24 months.

		Time in Months											
Phase	Task	2	4	6	8	10	12	14	16	18	20	22	24
n/a	Updating the literature review	x	x	x	x	x	x	x	x	x	x	x	x
2	Developing direct solution representation	x											
2	Developing indirect and hybrid solution representations	x	x										
3	Investigating unconstrained MO optimisation		x	x	x								
3	Extending MO approach to handle SLA constraints					x							
4	Designing and implementing semantic selection technique					x	x	x					
5	Extending planning/EC approach to handle QoS changes							x	x				
5	Creating a strategy for handling service failure								x	x			
6	Writing the first draft of the thesis										x	x	
6	Editing the final draft											x	x

been completed. The second phase, which corresponds to the first objective of the thesis, is currently in progress and is expected to be finished on time, thus allowing the remaining phases to also be carried out as planned.

4.3 Project Timeline

The phases included in the plan above are estimated to be completed following the timeline shown in Table 4.2, which will serve as a guide throughout this project. Note that part of the first phase has already been done, therefore the timeline only shows the estimated remaining time for its full completion.

4.4 Thesis Outline

The completed thesis will be organised into the following chapters:

- *Chapter 1: Introduction*
  This chapter will introduce the thesis, providing a problem statement and motivations, defining research goals and contributions, and outlining the structure of this dissertation.

- *Chapter 2: Literature Review*
  The literature review will examine the existing work on Web service composition, discussing the fundamental concepts in this field in order to provide the reader with the necessary background. Multiple sections will then follow, each of them analysing the problem from a different angle, considering issues such as QoS-aware composition, semantic selection, and composition methods for dynamic environments. The focus of this review is on investigating composition techniques that are based on Evolutionary Computation and on AI planning.

- *Chapter 3: A Hybrid Planning/EC Approach to Web Service Composition*
  This chapter will introduce a new approach to Web service composition that combines elements of AI planning with Evolutionary Computation techniques. One of the critical aspects of this new approach is the representation of composition candidates, therefore multiple representation possibilities will be analysed and compared to identify the most suitable model.

- *Chapter 4: Many-Objective Optimisation of Compositions with Multiple Paths*
  In this chapter, many-objective optimisation techniques will be employed to allow each quality dimension of compositions to be improved independently. After investigating these techniques in an unconstrained manner, SLA constraints will also be considered. While these constraints add to the complexity of the problem, they may also prove useful when filtering potential solutions, thus experiments comparing the constrained and unconstrained approaches will be conducted.

- *Chapter 5: Semantic Web Service Selection in a Planning-Based Composition Technique*
  A novel semantic approach for selecting which services are to be included in the composition will be proposed in this chapter. This approach focuses on selecting candidate services to be used in a planning-based composition technique without relying on precalculated semantic distances, as existing approaches have done. A comparison will be performed between this novel approach and the those that rely on precalculations.

- *Chapter 6: EC-Based Compositions in a Dynamic Environment*
  This chapter will extend the composition technique discussed throughout this thesis to work in dynamic environments. Firstly, a way of re-optimising solutions as quality values fluctuate will be proposed, and then an approach for responding to service faults will be discussed. The extended technique will be compared against the existing dynamic composition approaches, which do not use EC techniques, to establish whether there are quality gains in the solutions produced.

- *Chapter 7: Conclusions and Future Work*
  In this chapter, conclusions will be drawn from the analysis and experiments conducted in the different phases of this research, and the main findings for each one of them will be summarised. Additionally, future research directions will be discussed.

4.5 Resources Required

4.5.1 Computing Resources

An experimental approach will be adopted in this research, entailing the execution of experiments that are likely to be computationally expensive. The ECS Grid computing facilities can be used to complete these experiments within reasonable time frames, thus meeting this requirement.

4.5.2 Library Resources

The majority of the material relevant to this research can be found online, using the university's electronic resources. Other works may either be acquired at the university's library, or by soliciting assistance from the Subject Librarian for the fields of engineering and computer science.

4.5.3 Conference Travel Grants

Publications to relevant venues in this field are expected throughout this project, therefore travel grants from the university are required for key conferences.

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] BANSAL, S., BANSAL, A., GUPTA, G., AND BLAKE, M. B. Generalized semantic web service composition. *Service Oriented Computing and Applications 10*, 2 (2016), 111–133.

[7] 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.

[8] 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.

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

[10] BOOTH, D., HAAS, H., MCCABE, F., NEWCOMER, E., CHAMPION, M., FERRIS, C., AND ORCHARD, D. Web services architecture. w3c working note. *W3C Working Notes* (2004).

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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.

[21] CURBERA, F., NAGY, W., AND WEERAWARANA, S. Web services: Why and how. In *Workshop on Object-Oriented Web Services-OOPSLA* (2001), vol. 2001.

[22] DA SILVA, A., MA, H., AND ZHANG, M. Graphevol: A graph evolution technique for web service composition. In *Database and Expert Systems Applications*, vol. 9262. Springer International Publishing, 2015, pp. 134–142.

[23] 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 *2014 IEEE Congress on Evolutionary Computation (CEC)* (2014), IEEE, pp. 3127–3134.

[24] DA SILVA, A. S., MA, H., AND ZHANG, M. A gp approach to qos-aware web service composition including conditional constraints. In *Evolutionary Computation (CEC), 2015 IEEE Congress on* (2015), IEEE, pp. 2113–2120.

[25] DA SILVA, A. S., MA, H., AND ZHANG, M. Genetic programming for qos-aware web service composition and selection. *Soft Computing* (2016), 1–17.

[26] DA SILVA, A. S., MEI, Y., MA, H., AND ZHANG, M. Particle swarm optimisation with sequence-like indirect representation for web service composition. In *European Conference on Evolutionary Computation in Combinatorial Optimization* (2016), Springer, pp. 202–218.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] FENSEL, D., FACCA, F. M., SIMPERL, E., AND TOMA, I. *Semantic web services*. Springer Science & Business Media, 2011.

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

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

[35] 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.

[36] 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.

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

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

[39] 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.

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

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

[42] 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).

[43] 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.

[44] 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.

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

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

[47] 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.

[48] 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.

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

[50] KOPECKỲ, J., VITVAR, T., BOURNEZ, C., AND FARRELL, J. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing 11*, 6 (2007).

[51] KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[52] LAUSEN, H., AND FARRELL, J. Semantic annotations for wsdl and xml schema. *W3C recommendation, W3C* (2007), 749–758.

[53] LAUSEN, H., POLLERES, A., AND ROMAN, D. W3c member submission-web service modeling ontology (wsmo). *W3C. Available at¡ URL: http://www. w3. org/Submission/WSMO* (2005).

[54] LÉCUÉ, F. Optimizing qos-aware semantic web service composition. In *International Semantic Web Conference* (2009), Springer, pp. 375–391.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] Lin, N., Kuter, U., and Sirin, E. *Web service composition with user preferences*. Springer, 2008.

[60] 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 *International Conference on Grid and Cooperative Computing* (2005), Springer, pp. 84–89.

[61] 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.

[62] Ma, H., Wang, A., and Zhang, M. A hybrid approach using genetic programming and greedy search for qos-aware web service composition. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XVIII*. Springer, 2015, pp. 180–205.

[63] 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.

[64] 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.

[65] 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.

[66] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al. Owl-s: Semantic markup for web services. *W3C member submission 22* (2004), 2007–04.

[67] 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.

[68] Mier, P. R., Pedrinaci, C., Lama, M., and Mucientes, M. An integrated semantic web service discovery and composition framework.

[69] Miller, J. F., and Thomson, P. Cartesian genetic programming. In *Genetic Programming*. Springer, 2000, pp. 121–132.

[70] 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.

[71] 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.

[72] 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.

[73] O'LEARY, D. Review: Ontologies: A silver bullet for knowledge management and electronic commerce. *The Computer Journal 48*, 4 (2005), 498–498.

[74] OVERDICK, H. The resource-oriented architecture. In *Services, 2007 IEEE Congress on* (2007), IEEE, pp. 340–347.

[75] PALIWAL, A. V., SHAFIQ, B., VAIDYA, J., XIONG, H., AND ADAM, N. Semantics-based automated service discovery. *IEEE Transactions on Services Computing 5*, 2 (2012), 260–275.

[76] PAOLUCCI, M., KAWAMURA, T., PAYNE, T. R., AND SYCARA, K. Semantic matching of web services capabilities. In *International Semantic Web Conference* (2002), Springer, pp. 333–347.

[77] PAPAZOGLOU, M. P. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on* (2003), IEEE, pp. 3–12.

[78] PAPAZOGLOU, M. T. P., dustdar, s., leymann, f.. service-oriented computing. research roadmap, 2006.

[79] 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.

[80] 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.

[81] PETRIE, C. J. *Web Service Composition*. Springer, 2016.

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

[83] 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.

[84] 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).

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] SHET, K., ACHARYA, U. D., ET AL. A new similarity measure for taxonomy based on edge counting. *arXiv preprint arXiv:1211.4709* (2012).

[90] SHI, Y., ET AL. Particle swarm optimization: developments, applications and resources. In *evolutionary computation, 2001. Proceedings of the 2001 Congress on* (2001), vol. 1, IEEE, pp. 81–86.

[91] 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.

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

[93] 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.

[94] 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.

[95] WADA, H., SUZUKI, J., YAMANO, Y., AND OBA, K. E^3: A multiobjective optimization framework for sla-aware service composition. *IEEE Transactions on Services Computing 5*, 3 (2012), 358–372.

[96] WANG, A., MA, H., AND ZHANG, M. Genetic programming with greedy search for web service composition. In *International Conference on Database and Expert Systems Applications* (2013), Springer, pp. 9–17.

[97] 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.

[98] 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.

[99] 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.

[100] 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.

[101] 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.

[102] 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.

[103] 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.

[104] 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.

[105] 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.

[106] 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).

[107] 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. In *2008 10th IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services* (2008), IEEE, pp. 347–350.

[108] 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.

[109] YU, Q., LIU, X., BOUGUETTAYA, A., AND MEDJAHED, B. Deploying and managing web services: issues, solutions, and directions. *The VLDB JournalThe International Journal on Very Large Data Bases 17*, 3 (2008), 537–572.

[110] ZENG, L., BENATALLAH, B., DUMAS, M., KALAGNANAM, J., AND SHENG, Q. Z. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web* (2003), ACM, pp. 411–421.

[111] 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.

[112] 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.

[113] 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.