The Pennsylvania State University
The Graduate School

EFFICIENT WEB SERVICE COMPOSITION: FROM SIGNATURE-LEVEL TO BEHAVIORAL DESCRIPTION-LEVEL

A Dissertation in
Computer Science and Engineering
by
Hyunyoung Kil

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2010

The dissertation of Hyunyoung Kil was reviewed and approved* by the following:

Dongwon Lee
Associate Professor of Information Sciences and Technology
Dissertation Advisor, Chair of Committee

Soundar Kumara
Professor of Industrial Engineering

Swarat Chaudhuri
Assistant Professor of Computer Science and Engineering

Ae Ja Yee
Associate Professor of Mathematics

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Web services are software systems designed to support machine-to-machine interoperation over the Web. Many researches have been carried out for web service standards, and these efforts have significantly improved functionalities of *Service Oriented Architecture (SOA)* significantly. However, there still remain a number of research challenges. One of the remaining challenges is the *web service composition (WSC)* problem, i.e., when a single web service does not satisfy a given requirement, one wants to automatically combine web services to satisfy the requirement entirely. In this dissertation, we tackle this WSC problem in three levels, i.e., a *signature level*, a *behavior description level* and a *QoS description level* based on web service descriptions.

First, for a signature-level approach where each web service is described by its signature in *WSDL*, we first analyze the topological landscape of a web service network formed by real-world web services. We then propose a *SAT-based* algorithm based on the analysis.

Second, for web services that provide behavioral descriptions in addition to signatures, we first define a realistic model for the WSC problem, and investigate the *computational complexities* for the composition of web services on restricted (i.e., with *full* observation) and general cases (i.e., with *partial* observation). We then prove that the WSC problem with full observation is *EXP-hard* and the WSC problem with partial observation is *2-EXP-hard*. To solve these high complexities, we also propose *approximation-based* algorithms using *abstraction* and *refinement*.

Third, the previous two approaches consider only functional requirements specified in WSDL or BPEL. However, non-functional ones, such as *Quality of Services* (*QoS*) constraints, help clients select a service provider with good quality. In this case, the main aim of the WSC problem is to find a composite web service which satisfies a given complicated task with the optimal QoS value, which is called *QoS-aware WSC* problem. We first propose to apply *anytime algorithm* based on beam stack search to the QoS-aware WSC problem. Moreover, to improve the basic anytime algorithm, we propose *dynamic beam width* with more heuristics, i.e., *short backtracking* and *upper bound propagation*.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First of all, I thank my advisor, Dr. Dongwon Lee, for offering me technical and moral support during the last six years. He has encouraged me to study various promising research problems, and trusted me with his generosity and patience. Without his guidance, inspiration and contribution, this dissertation would not be achieved.

Second, I would like to express my thanks to my committee comprised of Prof. Kumara, Prof. Swarat, and Prof. Yee for their great service on my thesis. With generous effort and time, they assisted me to accomplish this dissertation. Their unstinted advice for my work be a great help to my future research.

I am also thankful to Prof. Wonhong Nam for his cooperation. While working at Penn State University as a research associate, I worked with him. His profound knowledge of symbolic model checking and controller synthesis was, without question, a great help for me to solve the problems.

Dr. Seog-Chan Oh introduced me the web service composition field and worked together for the first problem. I want to express thanks to him for his cooperation.

Also special thanks go to all the members of the Pike group. Their friendships and passions for the research always encourage me during my whole Ph.D program.

My allegiance to computer science is mainly due to the excellent education I received at Korea University. I am thankful to the faculty on its Computer Science and Engineering Department, especially, Prof. Chong Sun Hwang, my advisor at a master program and Prof. Hae-Chang Rim, my advisor at a bachelor program.

Also this thesis would not exist without the love and support of my family, especially, my parents and my husband. Sacrifice and diligence of my parents influence me as the most priceless lesson, and my husband always stands by me and cheers me up to let me get through the difficulties.

I would also like to use this opportunity to thank all my friends on Penn State campus. Because of them, my stay at State College has been very enjoyable and memorable.

Dedication

To My Mom in Heaven with all my love and respect.

Chapter 1

Introduction

The *World Wide Web (WWW)* which began as a system for exchanging research information between scientists, is now a part of daily life for billions people. As it became a new medium of communication for all classes of the people, the data and applications available on the web got to be tremendous. Since its spectacular growth went beyond the prediction of its innovators, it is no wonder that people cannot be satisfied with the web as it stands. As a result, a new web paradigm for better use of WWW has been suggested, which is called *semantic web*.

The semantic web is an extension of the existing web as a platform where people can define not only information but also the meaning of information through meta data. This technology makes it possible for machines to comprehend the web content. Based on this understanding, machines can perform more than the current tedious works, e.g., automatically finding, sharing, and combining information on the web.

One of key players in enabling the semantic web is a *web service.* web services are software systems designed to support machine-to-machine inter-operation over the web. Examples of web services span several application domains including public internet applications (e.g., Google search) and e-commerce (e.g., stock trading). Many researches have been carried out for the web service standards, e.g., *WSDL (Web Service Description Language)* [1], *SOAP (Simple Object Access Protocol)* [2] and *UDDI (Universal Description Discovery and Integration of web services)* [3]. These efforts have significantly improved flexible and dynamic functionality of *Service Oriented Architecture (SOA)* in the current web services. However, abundant research challenges still remain; e.g., automatic web service discovery, web service composition, and formal verification of composed web services. Given a set of web services and a user request, a *web service discovery problem* is to automatically find a web service satisfying the request. Often the client request cannot, however, be fulfilled by a single existing service. For example, a client wants to reserve both a flight ticket and a hotel room but there are two separate web services for flight ticket reservation and hotel reservation. In this case, one wants to automatically combine web services to satisfy the requirement entirely, which is called *web service composition (WSC)* problem. Service composition techniques emerge from the technology of choice for building applications over distributed systems [4, 5]. The main characteristics of web services, i.e., program accessability and loosely-coupled relationships, motivate automatic web service composition to be a practical solution. Since seamless composition of web services has enormous potential in streamlining business-to-business application integration, this WSC problem has received much interest from industry as well as academic research.

Web service composition depends on which information on the web service can be used for composition. In this dissertation, we investigate this WSC problem in three different service descriptions, e.g, a *signature level*, a *behavior description level*, and a *QoS description level*.

In a signature level, a web service is specified in WSDL as the current web services stands. The WSDL includes a service operation signature which provides a syntax of the message, including an invoked operation name with each set of input and output parameters. Given a set of web services and a desired output data, the web service composition based on WSDL signatures is to find out a *sequence of web services* which finally generated the desired outputs by *parameter matching*. In spite of its applicability to the real-world, there has been little study as to how useful current public web services are. In particular, since web services that are specified in WSDL and published in UDDI form a network, one can apply network analysis methods to study the characteristics of web service networks. Therefore, as starting the signature level approach, we first analyze the topological landscape of the web service networks formed by real-world web services. In general, the topological structure of a network affects the processes and behaviors occurring in the network. Accordingly, understanding the structural properties of networks often help gain important insights and develop efficient algorithms [6, 7]. To investigate properties of the current web services, we collect a number of public web services from public repositories and by using Google, and then propose a flexible framework where different matchmakings (from strict syntactic to approximate matchmaking methods) can be incorporated in a unified manner. On the framework, the constructed web service networks with diverse granularity show the small world and power-low-like properties. These findings provide a valuable hint to web service composition techniques, which allows the techniques to focus on short paths and hub nodes first. Based on these observations, we propose a web service composition technique to find an optimal composition. Given a set of web services and a requirement web services described in WSDL, our algorithm identifies the shortest sequence of web services such that we can legally invoke the next web service in each step and achieve the desired requirement eventually. We first reduce the composition problem into a *reachability problem* on a *state-transition system* where the shortest path from the initial state to a goal state corresponds to the shortest sequence of web services. To solve the reachability problem, we employ a state-of-the-art SAT solver, SATzilla [8]. It is the first work to employ SAT solvers for this problem setting even though several approaches [9, 10] use AI planning techniques for web service composition.

In the signature level approach, however, users can simply invoke the sequence of web services computed, but they cannot react exactly to the output values returned from web services in the sequence during runtime. Accordingly, researches suggest other specification languages including additional information such as functional annotations of how it behaves. As a next step, we study web service composition problem on the behavioral description level.

The behavioral description languages such as *Web Service Business Process Execution Language (WS-BPEL)* [11] and *OWL-S* [12] have more prominent status in the service composition area. Note that since OWL-S, a well-known semantic web language, is also able to describe how and what a service process does as *process model*, it can be considered as one of the behavior description languages. These languages specify service behaviors and interactions with other services as a sequence of activities. The behaviors of web services are specified as a state-transition system. That means, it is formally described that on receiving a specific input, what values for outputs a web service will return and what state it will proceed to. The composition with this information computes a strategy suggesting the order of web services—for a certain output value, which web service should be invoked in next with a certain input value in order to satisfy a user provided goal. In general, the strategy to guarantee achieving the goal can be represented as a state-transition system called a *coordinator*. The behavioral description-based web service composition makes more useful suggestion that can be executed in runtime. Therefore, given a set of behavioral descriptions of web services and a goal, web service composition problem is to synthesize a *coordinator web service* that controls the set of web services to satisfy the goal.

For WSC problem on behavioral descriptions, we first define a realistic model for the web service composition problem on behavioral descriptions with a state-transition system, and investigate the computational complexities for web services composition problem on restricted (i.e., with full observation) and general cases (i.e., with partial observation); EXP-hard and 2-EXP-hard, respectively. To the best of our knowledge, no study has investigated the computational complexity (i.e., lower bound) of the WSC problem with complete proofs although the investigation for the complexity can provide better insights to precisely understand the WSC problem, to know what is possible, and to identify interesting sub-problems.

These results suggest studying efficient approximation solutions to the WSC problem. Toward this challenge, we propose two approximation-based algorithms using abstraction and refinement. To the best of our knowledge, it is the first attempt to apply an abstraction technique to the WSC problem. Even, in *planning under partial observation* which has a strong connection with the WSC problem, no study has attempted to apply abstraction techniques. The first abstraction step is to reduce the original web services to the abstract ones with less variables, and we then try to solve the abstract problem. If we identify a coordinator that controls the abstract web services to satisfy a given goal, than the coordinator can control the original web services to satisfy the goal since the abstract web services over-approximate the concrete ones. Otherwise, we refine the abstract web services by adding variables, and repeat to find a solution. For abstraction, we propose two methods—*signature-preserving abstraction* and *signature-subsuming abstraction*. We report on the performance of our tool on realistic problems comparing with a basic algorithm without abstraction/refinement.

Whereas the previous two approaches focus to satisfy only functional requirements specified WSDL or BPEL, the QoS description level approach considers non-functional ones, such as *Quality of Services* (*QoS*) constraints. In this case, the aim of the WSC problem is to find a composite web service which satisfies a given complicated task with the optimal accumulated QoS value, which is called *QoS-aware WSC* problem. As a natural price of a web service, QoS such as response time and throughput helps clients to select a service provider with good quality. Therefore, recently the QoS-aware WSC problem has received attention of researchers, and several new standards are introduced (e.g., *WS-Policy* [13], the *Web Service Level Agreement (WSLA) language* [14] and *WS-Agreement* [15] for the specification of QoS requirements and service-level agreements of a single web service). When a large scaled problem instance is given, this problem is intractable since it is reduced to a global optimization problem. However, many engineering tasks and users in the real world require real-time responsiveness, even in such a large size of a given set of web services.

Therefore, we propose applying *anytime algorithm* [16] to the QoS-aware WSC problem. While traditional algorithms provide only one answer after completely terminating a fixed number of computations, anytime algorithms can quickly return a set of approximate answers to any given input and the quality of their best-so-far answers improves along with execution time. By using an anytime algorithm for the QoS-aware WSC problem, we can identify a composite web service with high quality earlier than optimal algorithms. To the best of our knowledge, there is no work to employ an anytime algorithm to the WSC problems.

Our anytime algorithm proposed in this dissertation is based on the *beam stack search* [17] which explores, in each level, a fixed number of candidate states called *beam width*. In this searching mechanism, a bad decision in early phases requires more searching space than a bad selection in late phases does. To reduce this cost, we propose to *dynamically* assign a larger beam width to early phases so that we can consider more qualified candidates. In addition, we propose two more heuristics, i.e., *short backtracking* and *upper bound propagation*, to improve the quality of current approximate solutions and overall execution time. We validate our proposal with experiment on a number of examples that are produced by the web service testset generator for web services challenge 2009 [18].

This dissertation is organized as follows. In Chapter 2, we introduce web services as background, which includes the definitions of web services and web service description languages. Chapter 3 presents a signature-based web service composition technique. We first analyze the topological structure of the web service networks and we propose a SAT-based algorithm based on our observations. Chapter 4 describes a behavior description-based web service composition. We define realistic models for this problem, and investigate the computational complexities. Then, we propose approximation-based algorithms using abstraction and refinement. Chapter 5 presents a QoS-aware web service composition. For this problem, we propose to apply an anytime algorithm based on beam stack search and then heuristics for improvement. In Chapter 6, we conclude this dissertation.

Chapter 2

Background

2.1 Preliminaries

2.1.1 Web Services

Web services are considered as one of vital building blocks for Semantic Web [19]. Indeed, the concrete definition of web services has evolved in recent years. "a business function made available via the Internet by a service provider, and accessible by clients that could be human users or software applications" [20] and "loosely coupled applications using open and cross-platform standards, which inter-operate across organizational and trust boundaries" [21] are examples of various definitions. The W3C Web Services Architecture Working Group defines a web service as "a software application identified by an URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts. A web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols" [22]. However, among different web service features which each definition emphasizes on their own view, the followings are considered as the main features of web services [23]:

- *Program accessibility*: Web services are designed to be invoked by other web services and applications, e.g., software programs on any platforms. By employing a set of XML standards to define and describe web service functionalities, client programs as well as human users can understand and utilize

Composition	BPEL
Publication and Discovery	UDDI
Service Description	WSDL
Transport Layer	HTTP, FTP, SMTP
Message Exchange	SOAP
Data Description	XML

Figure 2.1. Overview of Web Service Standards

the service better. Then, more complicated tasks such as discovery and composition of web services can be facilitated or even automated.

- *Loosely-coupled relationships*: Any kind of data can be exchanged between web services (e.g., (semi-)structured, textual) as long as it is embedded in an XML-based formats. The use of XML-based messaging communication model caters for loosely coupled relationships in the contrast with other distributed computing technologies (like CORBA, RMI, EJB, etc.) which generally yield tight-coupled systems by using object-based communication. While the latter can be more suitable for intra-enterprise environments, the technical features of web services make them more reusable and thus more appropriate for inter-enterprise and global environments.

Both features give an advantage to automatic service composition over the Web and stimulate companies to support web services for their enterprise applications.

Figure 2.1 shows the main web service technology standards, all based on XML. A web service interface is described in WSDL and web services exchange messages encoded in SOAP transported over HTTP or other Internet protocols such as FTP and SMTP. These specifications of web services are published in UDDI, a registry of web services, for a client (program) to find out an appropriate service. In addition, if needs arise, service composition can be specified as a flow of activities in BPEL.

Based on these standards, a typical web service life-cycle envisions the following scenario (see Figure 2.2): a service provider publishes the WSDL description of its service in UDDI. Subsequently, service requesters can inspect UDDI and locate/discover web services that are of interest. Using the information provided by the WSDL description, they can directly invoke the corresponding web service. By

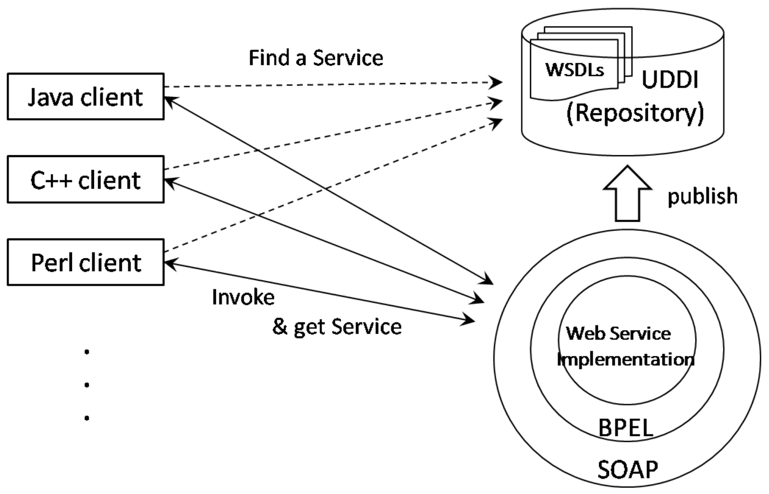


Figure 2.2. Mechanism of Web service

relying on these standards, web services hide any implementation details, therefore increasing cross-language and cross-platform interoperability.

In the following sections, we will introduce WSDL and BPEL as examples of web service description languages.

2.1.2 Web Services in WSDL

As an XML-based language, the *Web Services Definition language* (*WSDL*) is machine processable, being a structured and standardized way to describe web interfaces of services. By interpreting XML tags, applications can understand what the service is and how to access to the service.

The WSDL document has two major parts, an *abstract interface* part and an *implementation* part. First, the abstract interface of the service specifies the *data types*, *messages* and *portTypes* with the corresponding operations. XML schema syntax is used to define platform-independent data types for messages to use. Messages of an operation are an abstract definition of the data being transmitted. Each message has a name and a set of parts of certain types which can be defined as any XML schema data type or the data type previously defined. Parts represent input/output parameters declared in input or output messages. Operations are grouped into port types. Port types describe abstract end points of a web service such as a logical address under which an operation can be invoked. The implementation part binds the abstract interface to concrete network protocols and message formats (e.g., SOAP, HTTP). The separation of two parts allows the reuse of these definitions.

The following elements in a WSDL document are used to define services:

- *Types*: a container for data type definitions using XML schema.
- *Message*: an abstract, typed definition of the data being communicated.
- *Operation*: an abstract description of an action supported by the service.
- *Port type*: an abstract group of operations supported by one or more endpoints.
- *Binding*: a concrete protocol and data format specification for a particular port type.
- *Port*: a single endpoint defined as a combination of a binding and a network address.
- *Service*: a collection of related endpoints.

A WSDL definition for a service operation is published over the Internet by its service provider. In this way, a client (program) can read it and determine what functions are available at the server for automating communications between applications (refer to Figure 2.2 in 2.1.1).

An example (see Figure 2.3) shows a part of the WSDL file associated with a service operation to find a restaurant name and its address given a zip-code and a food preference. It includes two messages: `findRestaurantRequest` message with two parameters, an integer `zipcode` and a string `foodPreference`, and `findRestaurantResponse` message with two parameters, a string `restaurantName` and a string `address`. They are an input and an output message for `findingRestaurant` operation which is one of operations in the portType named `RestaurantServicePortType`. The last part including `<binding>` and `<service>` tags show that the `findingRestaurant` operation and its portType are binded to `RestaurantServiceSoapBinding` and `RestaurantService`.

After reading this WSDL definition, a client program wishing to get the name and address of a restaurant may invoke `findRestaurant(16801, ‘‘Thai’’)` operation to the web service.

```
<wsdl:definitions targetNamespace="http://..." xmlns="http://schemas.xmlsoap.org/wsdl/">
   <wsdl:message name="findRestaurantRequest">
      <wsdl:part name="zipcode" type=‘‘xsd:integer"/>
      <wsdl:part name="foodPreference" type=‘‘xsd:string"/>
   </wsdl:message>
   <wsdl:message name="findRestaurantResponse">
      <wsdl:part name="restaurantName" type=‘‘xsd:string"/>
      <wsdl:part name="address" type=‘‘xsd:string"/>
   </wsdl:message>
     ...

   <wsdl:portType name=‘‘RestaurantServicePortType">
      <wsdl:operation name=‘‘findingRestaurant">
         <wsdl:input message=‘‘findRestaurantRequest"/>
         <wsdl:output message=‘‘findRestaurantResponse"/>
      </wsdl:operation>
       ...

   </wsdl:portType>
     ...

   <wsdl:binding name=‘‘RestaurantServiceSoapBinding" type=‘‘ResrraurantServicePortType">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
         <wsdl:operation name=‘‘findingRestaurant"/>
            <soap:operation soapAction="" style="document"/>
       ...

   </wsdl:binding>

   <wsdl:service name=‘‘RestaurantService">
      <wsdl:port name=‘‘RestaurantServicePort" binding=‘‘RestaurantServiceSoapBinding">
       ...

   </wsdl:service>
</wsdl:definitions>
```

Figure 2.3. WSDL code for finding Reservation web service

2.1.3 Web Services in BPEL

Business Process Execution Language (*BPEL*: short for Web Services Business Process Execution Language (*WS-BPEL*)) is an executable language for specifying interactions with business processes. An example of business process might be a scenario where a company purchases some products from other company. The interactions between business processes can be described in two ways, an *executable business process* and a *business protocol* between processes. An executable business process models the actual behavior of the process as a participant in a business interaction. A business protocol uses process descriptions to specify the public message exchange behaviors of each of the parties involved in the protocol, without revealing their internal implementations. The processes involved in a business protocol are called *abstract processes*. A abstract process can be specified by simply coupling web service interface definitions. BPEL is used to define both kinds of processes, i.e., an executable business process and an abstract process.

BPEL depends on the following XML-based specifications: *WSDL*, *XML Schema*, *XPath* and *BPEL4WS*. The BPEL process model is layered on top of the service model defined by WSDL. That means, the BPEL4WS process model describes behaviors and interactions between services through their WSDL interfaces, e.g., portTypes and operations in a WSDL document. In addition, it supports XML schema and XPath to describe data types and links.

BPEL defines a process with a `<process>` tag including four basic sections:

- `<partnerLinks>` section defines the different parties that interact with the business process in the course of processing the order. Each partner link is characterized by a partner link type and a role name. This information identifies the functionality that must be provided by the business process and by the partner service for the relationship to succeed.

- `<variables>` section defines the data variables used by the process, providing their definitions using WSDL message types, XML schema simple types, or XML schema elements. Variables allow processes to maintain state data and process history based on messages exchanged.

- The remaining part of the process definition is defined by the outer `<sequence>` element, which contains the description of the normal process behavior for handling a request. In general, it involves three activities performed in order. The customer request is received (`<receive>` element), then processed (inside a `<flow>` section that enables concurrent behavior), and a reply message with the final approval status of the request is sent back to the customer (`<reply>`). This main processing section supports structured-programming constructs including `if-then-else`, `while`, `sequence` (to enable executing commands in order) and `flow` (to enable executing commands in parallel).

- `<faultHandlers>` section contains fault handlers defining the activities that must be performed in response to faults resulting from the invocation of the assessment and approval services.

```
<?xml version="1.0" encoding="UTF-8"?>
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/" .. >
    <import importType="http://schemas.xmlsoap.org/wsdl/" location="restaurantService.wsdl"
                                namespace="http://XXX.com/restaurant/wsdl/restaurantService/"/>

<partnerLinks>
   <partnerLink myRole="RestaurantService" name="ServiceProvider"
                                                        partnerLinkType="restaurantServiceLT" />
       <partnerLink name="client" partnerLinkType="ServiceClientLT"
                                                         partnerRole="restaurantServiceClient"/>
    </partnerLinks>

<variables>
        <variable name="Request" messageType="findRestaurantRequest" />
        <variable name="Response" messageType="findRestaurantResponse" />
    </variables>

    <sequence>
      <receive name="acceptClientRequest" operation="receiveRequest"
             partnerLink="ServiceClientLT" portType="ServiceClientPT" variable="serviceRequest">
      </receive>

     <assign>
        <copy>
           <from> serviceRequest </from>
           <to> Request </to>
        </copy>
         ...
     <assign>

     <invoke partnerLink="RestaurantService" operation="findingRestaurant" inputVariable="Request"
      outputvariable="Response" />
     <reply operation="returnReply" partnerLink="ServiceClientLT" portType="ServiceClientPT"
                                                                           variable="Response">
     </reply>
  </sequence>

</process>
```

Figure 2.4. BPEL code for a finding restaurant web service

An example (see Figure 2.4) describes process to find restaurant in BPEL, using the WSDL example in Section 2.1.2. While the WSDL code describe only the service interface including function name with input/output and protocols , BPEL models the service process which handle a request for the operation, exploiting definitions in WSDL, e.g., message type, port types and operation name. In this example, after receiving a request for finding a restaurant service via a `client` partnerLink, the process assigns the input message (`serviceRequest`) to its variable (`Request`), invokes `findingRestaurant` operation whose interface is defined in WSDL and then replies the result of the operation back. Since BPEL specifies process activities as a sequence of working statements (as shown in this example), it also describe a composite service well.

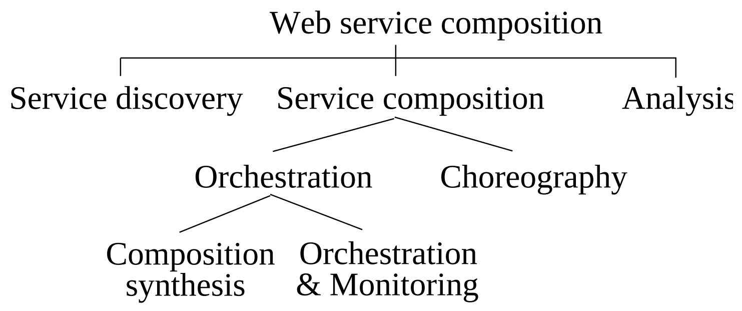


Figure 2.5. The activities of web service composition problem.

2.1.4 Web Service Composition

Web service composition (wsc) is required when a client request cannot be fulfilled by a single pre-existing web service. In such a case, one prefer integrating existing web services to satisfy the request. In the work [4], this new value-added service and a process to generate the service are named as a *composite service* and *composition*, respectively, and then key activities of wsc problem are illustrated like Figure 2.5.

At first, *Service Discovery* is to find out one or a set of web service appropriate to a given request. How to locate a web service is also an inseparable issue in the discovery. For this, many researchers have focused on a centralized UDDI registry as an effective method to solve a web service discovery problem. UDDI, a standard for centralized repositories, stores information describing web services produced by service providers, and is queried by service requesters. As an alternative of UDDI, specialized portals, such as XMethods [24], BindingPoint [25], RemoteMethods [26], or eSynaps [27] appear. They gather web services via the manual registration and support a keyword-based service search by focused crawlers. However, as the number of web services grows, such a centralized approach quickly becomes impractical. As a result, systems building on ontology and/or using Peer-to-Peer (P2P) technologies have been introduced (e.g., Hypercube ontology-based P2P system [28] and the Speed-R [29]).

Next, *Service Composition* integrates the existing services for an given request. Since various research fields tackle the composition problem with their own flavors, the vocabulary representing the design issues in the composition is mixed in many literatures. Here, we identify these terms for general understanding of the composition issues:

- *Orchestration* vs. *Choreography*: Both of orchestration and choreography describe how web services can interact at the message level, including the business logic and execution order of the interactions. However, orchestration methods aim at synthesizing a new web service called a mediator which has a specialized role of controlling the other services by properly exchanging messages. WS-BPEL [11] is an example of a standard language for orchestration. A choreography, on the other hand, is more collaborative in nature in that each party involved in the process describes its own role for one shared goal. That means, the execution of the composition is distributed to all participating web services. As a result, a specification generated by the orchestration identifies the execution steps for the participating services while a choreography specification describes the set of allowable conversation for a composite web service. Their difference in the topology of the composite service clearly explains why orchestration methods are called centralized or mediated-based approaches whereas choreography methods are called distributed or peer-to-peer.

- *Composition synthesis* vs. *Orchestration*: Orchestration is frequently used with a limited meaning by separating a composition synthesis in literatures [30, 4, 31, 32]. Composition synthesis concerns how to generate a specification of how to coordinate the participating services. Whereas, orchestration here takes into account how to coordinate the various participating services by executing the specification generated by the composition synthesis. It also includes the functionality to monitor control and data-flow among the participants for the correct execution of the composite service.

  Note that in this dissertation, we deal with this composition synthesis. However, the service composition is more generally used as a representative, we use *a web service composition problem* instead of a composition synthesis in this thesis.

In addition, there are other issues that give an influence to the design of a composite service.

Both *static* and *dynamic* composition show the clear difference in service selection timing. In the static composition method, we can decide the services to be

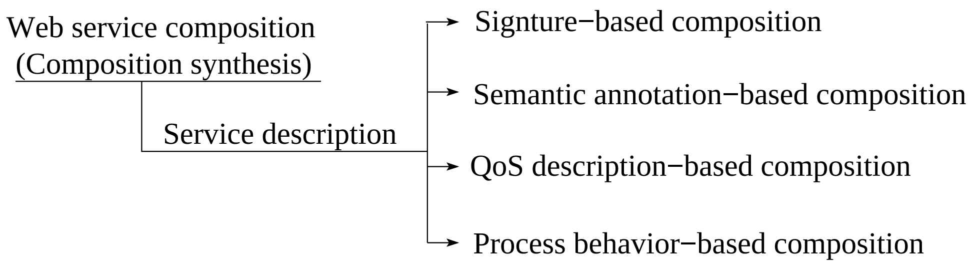

Web service composition
(Composition synthesis)
Service description
Signture−based composition
Semantic annotation−based composition
QoS description−based composition
Process behavior−based composition

Figure 2.6. The classification of web service composition problem.

composed at design time. Whereas, in the dynamic composition, it can happen at run time.

Finally, *Analysis* (*Verification*) in Figure 2.5 is particularly necessary because, by using automatic algorithms, a composite service is to be created from pre-existing services. The ultimate goal is to ensure that the eventual execution of a composite service produces the desired behavior. Ideally, one would be able to statically verify properties (e.g., in temporal logic) for composite services. There have been various attempts at developing such analysis methods for web services and workflow systems.

2.2 Related Work

2.2.1 Overview

There have been various approaches to address the web service composition (WSC) problem. In this chapter, we present an overview of the existing research on web service composition based on the web service description.

From an external point of view, service description with conceptual representation is one of significant features of web services to make realize the program accessability. By reading service specifications based on a set of XML standards, client programs as well as human clients can know what web services are and what task they can accomplish with the web services. Indeed, it is not easy for the programs to correctly understand the other services and make a decision without human. However, since we can develop models and algorithms for a new composite service on the basis of given specifications of web, how to describe a web service has been a continuing issue to the researchers. services.

Figure 2.6 describes our classification of related works on the WSC problem based on the service description aspect, i.e., *signature*, *semantic annotation*, *QoS description* and *process behavior* based compositions. We first shortly explain each grouping depending on each one.

- *Signature-based composition*

  This composition method assumes that a web service is simply expressed in terms of a syntactic interface of an operation (i.e., operation name and inputs/outputs), or possibly pre- and post-conditions. It lacks a formal semantics, e.g., a process behavior, communications of processes, or annotations of client requests and service functions, for automating the use of web service. Such WSDL like languages are the example. As a result, this kind of composition considers each service as a black box and returns a sequence of web services where inputs of a service may depend on the output returned by the previous service. At each point of the selection of next web service, the future is uniquely determined on the basis of the properties expressed in the client's request. Therefore, the client cannot do anything during the computation or execution of the composite service.

  The example (Figure 2.3 in Section 2.1.1) shows the instance of signature-based composition; given a zip-code and a food preference, a composer return a sequence of web service operations, a web service `findRestaurant` and a web service `findMap` to get a restaurant name and its address based on syntactic interfaces of operations.

- *Semantic annotation-based composition*

  Semantic web community advocates this approach which mainly relies on the specification of semantics of operations, explicit specification of goals of composition, pre- and post-conditions of the composed services in a common service ontology. An ontology which is included in semantic web service languages such as OWL-S enables the definition of web service content vocabulary in terms of object and complex relationships between them [33]. Figure 2.7 shows the example of ontology-based web service description. Based on the included ontology, composition algorithms can find out the web service

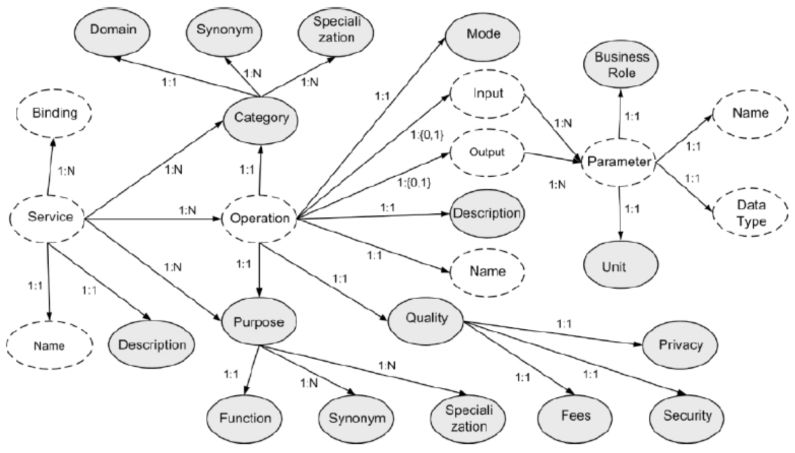


Figure 2.7. The example of ontology-based web service description.

which shows more accurate matching to the specification of the request than signature-based composition. Interestingly, OWL-S ontology includes both service profile (i.e., what a web service does) and service process model (i.e., how a web service behaves). However, the semantic web community takes into account mainly service profile, not service process model. Therefore, we divide the works in OWL-S into the semantic annotation-based approach and the behavior description-base approach separately.

- *QoS description-based composition*

  Since the number of web services on the Web increases rapidly, we can find several services which are functionally equivalent and thus can substitute each other. The selection among them can be done by non-functional properties, referred to as *Quality of Service* (*QoS*) attributes with each user's different preferences. QoS may be defined in terms of attributes such as price, response time, availability, reputation. When a user wants to consider constraints on the values of given QoS attributes, the composition problem is not a reachability checking problem but a global optimization problem. If a client wants the fast service processing, the composition algorithm should find a composite web service which achieves a given complicate task with the smallest total response time (Example in Section 5.1.1). Generally, this problem is computationally hard since it should compare all possible combinations of candidate web services.

- *Process behavior-based composition*

  A behavioral description of a web service process is a formal specification on what the web service executes internally and externally while interacting with users; e.g., describing what output value it returns for a given input and its state, and how it changes its internal state. The behaviors of web services represented as a state-transition system formally describes that on receiving a specific input, what output values a web service will return and what state it will proceed to. The composition with this information generates a strategy to guarantee achieving the goal can be represented as a state-transition system called a *coordinator*. For example, for the web service to find out a restaurant name and its address, rather than a simple sequence of web services in a signature-based composition, a process behavior-based composer can return a state-transition system specifying that for a specific zip code and a food preference of a client web service, how internal states of web services, `findRestaurant` and `findMap` change and what output values is computed based on the input values, and how the web services communicates with.

From the next, we see the related work based on the service descriptions, respectively. In Table 2.1, we summarize the related work.

2.2.2 Signature-based Composition

There are many researches based on signature description of web service [34, 35, 36, 37, 38, 39, 40, 41, 42]. In the work [34], Srivas et al. indicate that most of the industrial solutions on web service composition problem describe mainly the signature of individual web service operation. They view a web service as a remote procedure call between which the message is described with simple syntax, not with any additional semantics. Because this problem setting is similar to a traditional work-flow scheduling problem, workflow generation techniques have been applied to composite web services. EFlow [35] is a platform for the specification, enactment and management of composite services by using a static workflow generation method. It models a composite service by a graph that defines the order of execution among the nodes in the process. However, this graph should be specified manually first. Casati et al. [35] propose a prototype of composite service definition language (CSDL) considering dynamic environment. Polymorphic Process Model (PPM) [36] uses a method combined with static and dynamic. In PPM, a subprocess reference which is an abstract functionality information of the subprocess helps building an abstract composite process model, and the reasoning based on a state machine modeling a service enable a run-time service composition.

In this kind of approaches, the matchmaking techniques make a significant role in deciding which operation should /and can be invoked. At the beginning, the keyword-matching supported by the category browsing of UDDI are introduced. However, the keyword based matching cannot fully capture real functions of web services. To address this limitation, many researchers have developed various methods to assess the similarity of web services for matchmaking. Paolucci et al. [37] proposes a solution-based on DAML-S for semantic matching between service advertisements and capabilities. However, their matching algorithm is limited to comparing inputs and outputs of the advertisement with inputs and outputs of the request. In the work [38], the algorithm checks syntactic features (input and output events of component services). Wu [39] suggested a matchmaking process based on a lightweight semantic comparison of signature specifications in WSDL by means of several assessment methods. Wang and Stroulia [40] assessed the similarity of the requirement description of the desired service with the available services via the semantic information retrieval method and a structure matching approach. A survey of modern matchmaking algorithms and their applications to the web service matching is available in [39]. In our work, instead of proposing a particular matchmaking method, we advocate a flexible framework that can accommodate a plethora of matchmaking schemes in a unified manner. Due to the availability, we choose Cosine and WordNet based matching schemes in [43, 44]. However, note that it is also possible to incorporate the aforementioned matching approaches such as [39, 40] in our framework, if the implementation is available. Dong et al. [41] suggested a web service search engine, Woogle, which has the web service similarity search capability. Woogle first clusters parameter names into semantically meaningful concepts, which are then used to compute the similarity between parameter or operation names.

2.2.3 Semantic Annotation-based Composition

This approach is advocated by Semantic Web community which vision is to make resources accessible by content as well as by keywords. They think, by giving the meaning to the used term, machines can understand high level described task and then do the exact work. As a result, machine reasoning techniques based on a specific ontology are mainly presented.

The work [45] propose an ontology-based framework for the automatic composition of web services. It uses composability rules which compare the syntactic and semantic features of web services to determine whether two services are composable. Sirin et al. [46] use contextual information to find matching services at each step of service composition. They further filter the set of matching services by using ontological reasoning on the semantic description of the services (in OWL-S) in addition to user inputs.

The METEOR-S (METEOR for Semantic web services - [47, 48, 49, 50]) project associates semantics to web services, covering most of context in the service, and exploits them in the entire web process life cycle encompassing semantic description/annotation, discovery, composition, and enactment of web services. It presents semantic annotation of web services, semantic discovery infrastructure for web services (MWSDI: METEOR-S Web Service Discovery Infrastructure), and semantic composition of web services (MWSCF: METEOR-S Web Service Composition Framework). Shivashanmugam et al. [50] propose semantic process templates to capture the semantic requirements of web process on MWSCF.

SMAPLAN [51] attempts to combine semantic matching consisting of domain-dependent and domain-independent ontologies with AI planning techniques to achieve web services composition. For composition based on semantics, Mrissa et al. [52] have studied a context-based mediation approach to solve semantic heterogeneities between composed web services.

DERI is a noticeable ongoing project titled Semantic Web enabled Web Services (SWWS). DERI researchers recognize that to use the full potential of web services and the technology around UDDI, WSDL and SOAP, it is indispensable to use semantics, since current technologies provide limited support for automating web service discovery, composition and execution. Important objectives of the SWWS initiative include providing a richer framework for web Service description and discovery, as well as, providing scalable web Service mediation middleware.

There are various matching techniques with semantic annotation of service. LARKS [53] defines five techniques for service matchmaking: context matching, profile comparison, similarity matching, signature matching, and constraint matching. Those techniques mostly compare service text descriptions, signatures (inputs and outputs), and logical constraints about inputs and outputs. The ATLAS matchmaker [54] defines two methods for comparing service capabilities described in DAML-S. The first method compares functional attributes to check whether advertisements support the required type of service or deliver sufficient quality of service. The second compares the functional capabilities of web services in terms of inputs and outputs. No evaluation study is presented to determine the effectiveness and speed of the ATLAS matchmaker. Maedche and Staab [55] provided multiple phase cross evaluation to assess the similarity between two different ontology. The work [56] describes the design of a service matchmaker that uses DAML-S-based ontology. It uses techniques from knowledge representation to match service capabilities. In particular, it defines a description logic (DL) reasoner; advertisements and requests are represented in DL notations.

Although the previously mentioned works include the specification of the semantic information including QoS attributes, they do not consider a value optimization of a specific attribute in the composition. This kind of works are introduced to 2.2.4.

2.2.4 QoS description-based Composition

Recently, non-functionality, e.g., QoS, -aware web service composition has gained the attention of many researchers [57, 58, 59, 60, 61, 62] since it offers interesting applications of search strategies with user's constraints specified in a goal. To manage the web service composition problem with the goal including the constraints expressed with functional and non-functional properties of the services, applicable QoS model should be considered.

The projects such as METEOR [59, 63] and CrossFlow [64] present comprehensive QoS models. METEOR considers most of activities of QoS, including composition, analyzing, predicting, and monitoring QoS of workflow processes. They use Integer programming (IP) solutions for QoS optimization with the assumption linearity of the constraints and of the objective function. In the work [59], Aggarwal et al. strengthen the need to deal with more general constraint criteria, such as service dependencies or user preferences. CrossFlow proposes the use of continuous-time Markov chain to estimate execution time and cost of a workflow instance. However, these works do not deal with the dynamic composition of services which is more challenging.

The best combination of concrete services which is selected at a run time from a view of QoS may change during execution due to the dynamic nature of the web services environment, therefore one important requirement is that a feasible solution should be found in a short time.

Zeng at al. focuses on dynamic and quality-driven web service composition. In [57, 58], they use a planning technique for the composition with a global optimized QoS value where linear programming techniques is applied to find the optimal selection of component services. [add more dynamic works] They also consider data quality management in cooperative information systems [58]. They essentially focus on the cost, response time, availability and reliability attributes, where logarithmic reductions are used for the multiplicative aggregation functions, and the model is claimed to be extensible with respect to other similarly behaving attributes.

Ardagna et al. [60, 61] extends the linear programming model to include local constraints. Linear programming methods are very effective when the size of the problem is small. However, these methods suffer from poor scalability due to the exponential time complexity of the applied search algorithms.

Yu et al. [62] propose heuristic algorithms that can be used to find a near-to-optimal solution more efficiently than exact solutions based on a broker system. Their goal of service selection is to maximize an application-specific utility function under the end-to-end QoS constraints. In addition, to obtain both of optimality and efficiency, Alrifai et al. [65] propose a solution that combines global optimization with local selection techniques. The proposed solution finds the optimal decomposition of global QoS constraints into local constraints by using mixed integer programming (MIP) and then select the best web services that satisfy these local constraints by distributed local selection method.

Other various techniques are applied. [66] present the method using GAs because GAs can easily handle any kind of constraint. Umeshwar et al. [67] proposes a QoS optimization algorithm for web services based on traditional database query optimization technique. It focuses on ordering the operations to participating web services where the total response time is minimized. Yu et al. propose a query algebra to generate Service execution plans [68].

2.2.5 Process Behavior-based Composition

Since the behaviors of web services can be formally specified as a specific structure with abstract notion of possible activities (e.g., state-transition system), this kind of approaches suggest various abstract service models which is efficient to their composition algorithm.

Beradi et al. starts with a very abstract model of web services, called Colombo, based on an abstract notion of activities [69]. However, basically, there is a finite alphabet of activity names, but no internal structure is modeled, that is, no input, output, or interaction with the world. In the work [70], they use a transition system to specify the internal process flow of a web service. In the most general case, these are potentially infinite trees, where each branch corresponds to a permitted sequencing of executions of the activities. For the theoretical results, they restrict attention to systems that can be specified as deterministic finite-state automata, where the activities act as the input alphabet (i.e., where the edges are labeled by activities). This model is called as the Roman model.

The work [71] present a composition method to apply logical inferencing techniques on pre-defined plan templates. The service capabilities are annotated in OWL-S and then manually translated into situation calculus and Golog. Golog, the logic programming language implemented over Prolog is used to instantiate a plan. A user submits the request to the system, expressed as a kind of generic ConGolog [72] procedure. An agent instantiates the user specification with the services by pruning the situation tree where each node denotes a snapshot of the service configuration at some point. As a result, it generates a sequence of web services which can be executed by ConGolog interpreter.

In the work [73], Narayanan and McIlraith use DAML-S ontology for describing the capabilities of web services and encode the service descriptions in a Petri Net formalism. They provide decision procedures for web service simulation, verification and composition.

Hierarchical Task Network (HTN) planning [10] allows the expression of aggregate behavior2. A planner takes a composite processes as an input, that is, as descriptions of how to compose a sequence of single step actions and then tries to find a collection of atomic processes instances which form an execution path for some top-level composite process. However, it lacks of the ability to express non-determinism and iterations in compound tasks.

The work [74] is interesting in the point of a peer-to-peer framework for modeling and analyzing the global behavior of service compositions. Services exchange messages according to a predefined communication topology, expressed as a set of channels among services where a sequence of exchanged messages is referred to as conversation between the services. In this framework, properties of conversations are modeled as mealy machines to characterize the behavior of services. Given a desired global behavior and a composition infrastructure, e.g., a set of channels, a set of services and a set of messages, it synthesizes the specification of a sequence of activities executed by the services such that their conversations are compliant with the specification expressed by the regular language.

The behavior of a service can be captured by a complete or partially (incompletely) specified description. Incomplete information can make the mediator not sure what the current state is and what the next state is after performing a specific action. Since many web services associate with private data of their own database, the existence of incompleteness is more realistic assumption. However, only a few use realistic models with incomplete information (partial observability).

Pistore et al. define web service compositions with incomplete information [75, 76, 77]. They present composition algorithms which take into account service behavior and complex goals. however, it seems to disregard semantics information, as the quality of the discovered composition mainly depends on the goodness of the given goal (expressed by means of the EaGLe language), rather than on the functional attributes of services. Based on these algorithms, they implement a tool (MBP) to find a plan that satisfies a given goal over a planning domain which are formalized as non-deterministic state transition system where an action

Table 2.1. Summary of the related works

Signature	Semantic annotation	QoS description	Process behavior
Srivas [34], Casati [35], Schuster [36], Paolucci [37], Mecella [38], Wu [39], Wang [40], Dong [41]	Medjahed [45, 55], Sirin [46], Cardoso[47, 48], Sivashanmugam [50], Akkiraju [51], Mrissa [52], Ankolenkar [54], Li [56]	Grefen [64], Zeng [57, 58], Aggarwal [59], Oldham [63], Ardagna [61, 60], Canfora [66], Yu, Q. [68], Alrifai [65]	Sirin [10], Berardi [69, 70], McIlraith [71], Giacomo [72], Narayanan [73], Hull [74], Pistro [75, 76, 77], Brogi [79]

is a transition that may bring the system from one state to a set of possible successor states. This state transition system combines all the transition systems corresponding to the available services which can be modeled in OWL-S. MBP also can deal with uncertainty on the state called a belief state. In [76], the authors extend the previous work in order to handle asynchronous, message-based interaction between the domain and the plan. However, they do not consider QoS properties or complexity.

Recently, Fan et al. [78] investigate the complexity of web service composition based on query rewriting using views, but they do not include a non-deterministic web service in their problem setting, which is more natural. Moreover, their study does not provide the complete proof.

In the work [79], Brogi discusses the potential advantages of exploiting behavioral information for service discovery and composition, and the cost of generating such information and to the needed trade-off between expressiveness and cost and value of analyzing such information. It is an interesting work to understand the requirement of service behavior representation in web service.

2.2.6 Other Related Research

The WSC problem studied in this dissertation is related to many other fields; e.g., game theory [80, 81], open system synthesis with temporal logics [82, 83, 84, 85], controller synthesis for discrete systems [86, 87], and AI planning [88, 89, 90, 91].

First, composition problems have a close connection to game theory. Since the coordinator cannot control the non-determinism of a given set of web services and has to decide inputs to web services only with outputs from them, this problem can be considered as a two-player incomplete information game where the coordinator wants to win over web services. Reif [80] has proved that the problem of determining the outcome of universal games from a given initial position is 2-EXP-hard. Thomas [81] has surveyed the algorithmic theory of infinite games, and its role in automatic program synthesis and verification.

For open systems, next, various controller synthesis problems have been studied with temporal logics [92, 93]. Most of them have considered linear time logics [92] and often involved dealing with incomplete information [83, 84]. For branching-time cases, Antoniotti [82] has studied the synthesis of memoryless controllers with maximal environments. Kupferman and Vardi [85] have studied the problem of open system synthesis under incomplete information, and proved that the problem with a specification in CTL (CTL*) [93] are EXP-complete (2-EXP-complete, respectively).

Tsitsiklis [86] showed that a special class of the centralized supervisory control problem under partial observation is decidable in polynomial time but a general case is computationally intractable. Rudie and Willems [87] generalized the results on computational complexity of the supervisory control problem given in [86]; the question of whether there exist decentralized controllers that ensure that closed-loop behavior precisely equals some prescribed desired behavior is decidable in polynomial time.

Finally, the WSC problem is related to AI planning under partial observation [88, 89, 90, 91]. Herzig et al. [88] have proposed a dynamic logic EDL for planning under partial observability. In [89], a fully automatic planning tool MBP has been developed for this setting based on belief-states. The complexity of planning under partial observability has been studied in [90]. Moffitt [91] has explored a means to both model and reason about partial observability within the scope of constraint based temporal reasoning.

On the other hand, several researches have been performed in planning using abstraction. Huang et al. [94] propose an algorithm to reduce observation variables for strong plans. This technique, however, cannot identify such a variable until a plan is constructed. Thus, it cannot be applied to our problem. Armano et al. [95]

employ abstraction techniques for a hierarchical planner. Smith et al. [96] present an abstraction technique to generate exponentially smaller POMDP. However, to the best of our knowledge, there is no study for WSC problems or planning on partial observation using abstraction and refinement.

The most commonly used method in signature-based service composition is to specify the process model in network (graph). Therefore, it is worthwhile investigating this network model with the existing network analysis techniques. For related works for network analysis, many empirical networks are well modeled by complex networks including the scale free and small world networks. The small world networks are generated by Watts-Strogatz model [97]. Albert et al. [98] proposed a set of different models for generating scale free networks, based on the growing process of the Internet and other empirical complex networks. Denning [99] surveyed various network laws with a focus on the power-law distribution and the scale free networks. In this dissertation, we apply the developed techniques to examine web services networks.

Chapter 3

Signature-based Web Service Composition

A *signature* (from Latin *signare*) is a handwritten depiction of someone's name or even a simple "X" that a person writes on documents as a proof of identity and intent. A signature in computer programming means an *interface* of an operation (In fact, rather than a operation, a *function* or a *method* is used often in computer programming languages). An operation is commonly identified by its unique signature which usually includes the operation name, the number and type of its parameters, and its return type. Hence, we can say that a signature is the smallest type of an operation.

A web service is specified in a WSDL document where its operation signatures are inscribed. An operation signature provides the way to invoke the operation including syntactic information such as the operation name, its input/output message name and type, and its protocols. Based on WSDL signatures, given a set of web services and a desired output, the web service composition is to find out a *sequence of web services* which finally generated the desired outputs, by using matching techniques.

As the first step, we study as to how useful the current public web services are based on the graph theory. In Section 3.2, we construct web service networks formed real-world web services (Section 3.2.2) and identify network properties (Section 3.2.3). To construct the networks, we propose a flexible framework for web service match in 3.2.2. Since currently there are not enough "semantic web services" specified in OWL-S or WSDL-S yet, our study is limited to ones in WSDL. In addition, because WSDL standard does not support additional semantic annotations such as pre/post conditions or world state changes, our matching can be done based on parameter matching. Based on our observations, we suggest a SAT-based algorithm which finds out an optimal composition, that is, the shortest sequence of the web services. We report on a preliminary implementation and experiment for our solution with 7 sample examples for the WSC'07 competition [100]

3.1 Preliminaries

We introduce the *Small world* and the *Power-law* network properties which are shown in many robust real-world complex systems.

3.1.1 Small World and Power-Law

In general, a network is called the *small world network* if it shows the properties of both *random* and *regular networks* [101].

Definition 1 (Random Network). *A* random network *consists of N nodes, where each pair of nodes is connected with the probability p. As a result, edges are randomly placed among a fixed set of nodes [102].*

Definition 2 (Regular Network). *A* regular network *consists of N nodes, where a node i is adjacent to nodes $[(i+j) \bmod N]$ and $[(i-j) \bmod N]$ for $1 \le j \le K$. If $K = N-1$, it becomes the complete N-node graph, where every node is adjacent to every other $N-1$ nodes [102].*

Random networks are characterized by their short average distances among reachable nodes. On the other hand, in regular networks, each node has highly clustered neighbor nodes, such that the connectivity between neighboring nodes is very high. Consequently, small world networks show both (1) highly clustered structure and (2) small average shortest distance. To measure the connectivity and short average distances, the following metrics are often computed.

- L: The **average shortest distance** (i.e., number of hops) between reachable pairs of vertices. $L(p)$ is defined as L of Watts-Strogatz graph [97] with probability p. L_{random} is identical to $L(1)$.

- C: The **average clustering coefficient**. For a node i with v_i neighbors, $C_i = \dfrac{2E_i}{v_i(v_i - 1)}$, where E_i is the number of edges between v_i neighbors of i. C is the average clustering coefficient C_i for a network. Again, C_p is defined as C of the Watts-Strogatz graph with the probability p. C_{random} is identical to $C(1)$.

- $Index_{SN}$: The **small world network index** is defined as: $Index_{SN} = \frac{|C_{actual} - C_{random}|}{|L_{actual} - L_{random}|}$, where C_{actual} and L_{actual} represent C and L of the measured network, respectively, and C_{random} and L_{random} represent C and L of the random graph with the same number of nodes and average number of edges per node as the measured network.

If a network has the small world properties, then its L and C shows: $C_{actual} \gg C_{random}$ and $L_{actual} \gtrsim L_{random}$. That is, the average clustering coefficient is much larger than that of a random network, while the average shortest distance is slightly larger than or similar to that of a random network. Therefore, the more distinct the small world properties of a network are, the bigger $Index_{SN}$ of the network becomes.

A *power-law* distribution often occurs in complex systems where a majority of nodes have very few connections, while a few nodes have a high degree of connections. Typical power-law function has the form of $y = Cx^{-\alpha}$, and is captured as a straight line in log-log plots. The existence of power-law distribution has been observed in many real and artificial networks such as power grid, WWW, or collaboration network, and believed to be one of signs of mature and robust networks.

3.2 Topological Analysis of Web Service Network

In general, the topological structure of a network affects the processes and behaviors occurring in the network. For instance, the topology of a social acquaintance network may affect the spread rate of gossip or disease. Similarly, the topology of the Internet is known to be correlated with the robustness of communication therein. Accordingly, understanding the structural properties of networks often help gain better insights and develop better algorithms [6, 7].

Since current public web services which are specified in WSDL and published in UDDI can form a network, we can use network analysis methods to investigate the characteristics of web service networks. Therefore, we construct the networks consisting of real-world web services (Sections 3.2.2), and identify network properties (e.g., the small world and power-low properties) from web service networks (Sections 3.2.3).

3.2.1 Web Service on Signature

A *web service* in a WSDL file can be viewed as a collection of *operations*, each of which in turn consists of input and output *parameters*. When an operation op has input parameters $IN = \{p_1, \cdots, p_n\}$ and output parameters $OUT = \{q_1, \cdots, q_m\}$, we denote the operation by $op(IN, OUT)$. Furthermore, each parameter is a pair of $(name, type)$. We denote the name and type of a parameter p by $p.name$ and $p.type$, respectively. In order to decide if an operation op_1 can invoke an operation op_2, one needs to check if the types of op_1 and op_2 are compatible or not. Since this can be done trivially, from here forward, we assume that types of operations are all compatible, and focus on other issues instead.

3.2.2 The Matchmaking Framework

3.2.2.1 Flexible Matching

A network consists of nodes and edges. In our framework, different entities can be used as nodes and edges in a unified manner. First, as nodes, we consider three kinds—*parameters*, *operations*, and *web services*—from finer to coarser granularity. Second, as edges, we use the notion of *parameter matching* and *operation invocation*. The connection of parameter nodes can be considered as an information production flow—a cuisine and a zip code are able to produce the name and address of a restaurant by a web service `findRestaurant`, and the address can be used to obtain the map of the restaurant by a web service `findMap`.

When the "meanings" of two parameters, p_1 and p_2, are interchangeable, in general, they are said to be "matching" each other. The simplest way to check this is if two parameters have the *same* name and type: $(p_1.name = p_2.name) \wedge (p_1.type = p_2.type)$. Since web services are designed and created in isolation, however, this naive matching is often too rigid and thus misses cases like p_1 = ("password", string) and p_2 = ("passwd", string). On the other hand, if web services are annotated with rich semantics (e.g., using RDF [103] or WSDL-S [104]), then the so-called "semantic" matching can be easily reasoned out. However, in practice, the majority of public web services does not have annotated semantics yet. In order to cover all the spectrum of matching, therefore, we propose a generic Boolean function, `match`(p_1, p_2), that determines if two parameters p_1 and p_2 are matching or not. Formally,

Definition 3 (type-match). *A boolean function,* ***`type-match`***$(p_1.type, p_2.type)$*, returns True if: (1)* $p_1.type = p_2.type$*, or (2)* $p_1.type$ *is derived from* $p_2.type$ *in a type hierarchy, or vice versa.*

Definition 4 (name-match). *A boolean function,* ***`name-match`***$(p_1.name, p_2.name, D, \theta)$*, returns True if the distance between* $p_1.name$ *and* $p_2.name$ *by a distance metric* D *is below the given threshold* θ*: i.e.,* $D(p_1.name, p_2.name) \leq \theta$*.*

For instance, `name-match`("password", "passwd", =, 1) represents the exact matching, and would return False since "password" ≠ "passwd"[1]. Similarly, by using string matching functions such as Edit distance or cosine metric, one can express the approximate name matching. For instance, `name-match`("password", "passwd", edit-distance, 3) would return True since two names of parameters have the edit distance of 2 (< 3). In our experimentation, we use three metric functions:

[1]For the equality checking (i.e. =), the threshold value other than 1 is not meaningful.

= (i.e., literal equality), cosine distance with TF/IDF weights [105], and WordNet [44] based approximate distance. Based on two boolean functions above, we define the generic parameter match function as follows:

Definition 5 (match). *A boolean function,* `match`(p_1, p_2, D, θ)*, returns True if: (1)* ***`name-match`***$(p_1.name, p_2.name, D, \theta)$ *= True, and (2)* ***`type-match`***$(p_1.type, p_2.type)$ *= True.*

Definition 6 (Parameter Matching). *When a boolean function,* `match`(p_1, p_2, D, θ)*, returns True, it is said that a parameter p_1 matches a parameter p_2: "$p_1 \sim p_2$".*

On the other hand, the connections of operation nodes, i.e. invocations between web service operations, represent machine to machine interoperation which is the main goal of web services. When we want a map of a restaurant but not holding input data for invoking to `findRestaurant` service, `findRestaurant` service can help us to invoke `findMap` service by its result (an address of a restaurant).

In order to invoke an operation op_1 with input parameters $IN = \{p_1, \cdots, p_n\}$, one may need to provide values for input parameters. When one can provide all input parameters, op_1 is said "fully invocable". When one can provide at least one input parameter, op_1 is said "partially invocable".

In fact, not all input parameters are required to invoke the function. Some of them are semantically optional. For instance, Google API has a search operation with several input parameters, but it can be invoked by null values for many of parameters. Similarly, consider a client program wishing to invoke an operation op_1 first and then have op_1 invoke another operation op_2 directly (i.e., a case of web service composition). In this case, if the output parameters of op_1 satisfy all input parameters of op_2, then op_1 "fully" invokes op_2, and if output parameters of op_1 satisfy some input parameters of op_2, then op_1 "partially" invokes op_2.

Definition 7 (Operation Invocation). *For two operations, $op_1(IN_1, OUT_1)$ and $op_2(IN_2, OUT_2)$,*

- *op_1 fully invokes op_2, denoted by "$op_1 \mapsto op_2$", if for every mandatory input parameter $p \in IN_2$, there exists an output parameter $q \in OUT_1$ such that $q \sim p$.*
- *op_1 partially invokes op_2, denoted by "$op_1 \dashrightarrow op_2$", if there exists a mandatory input parameter $p \in IN_2$ and an output parameter $q \in OUT_1$ such that $q \sim p$.*

We denote each invocation by **full-** and **partial-invocation**, respectively.

3.2.2.2 Web Service Network Model

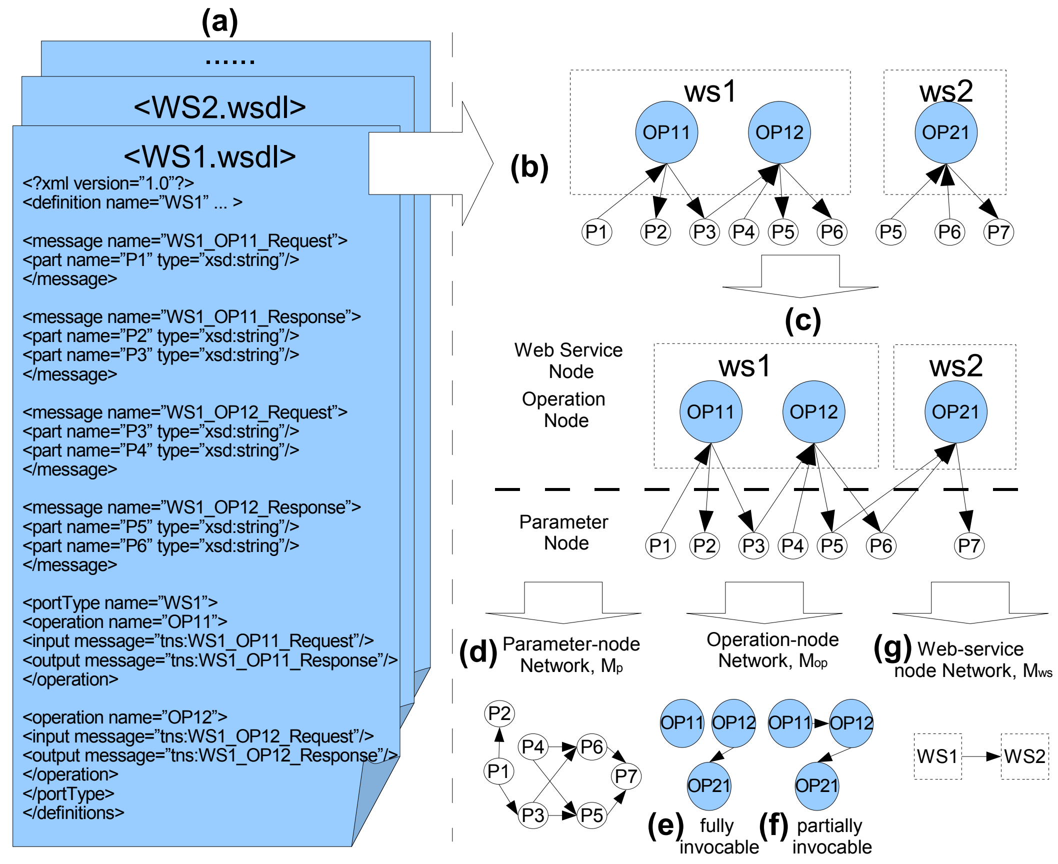
In this section, using the notions of nodes and edges defined in Section 3.2.2.1, we propose a flexible web service network model as follows.

Definition 8 (Web Service Network Model). *A web service network is generated by a 4-tuple model $\mathcal{M} = (T, D, \theta, I)$, where:*

- *T is the type of nodes and can be either "`p`" for parameters, "`op`" for operations, or "`ws`" for web services.*
- *D (and θ) is the distance metric to be used in parameter matching (and its threshold resp.).*
- *I is the type of operation invocation and can be either "`FI`" for full invocation (denoted as $\mathcal{M}^f$) or "`PI`" for partial invocation (denoted as $\mathcal{M}^p$).*

Example 1. *A model $\mathcal{M}_1$ = (`op`, cosine, 0.75, `FI`) generates an operation node network where parameter matching is done using cosine metric with a threshold 0.75. Furthermore, an edge from an operation op_1 to op_2 is added only when $op_1 \mapsto op_2$. On the other hand, another model $\mathcal{M}_2$ = (`ws`, word-net, 0.9, `PI`) generates a web service node network where intra-parameter matching is done using WordNet based approximate distance metric with a threshold 0.9. Furthermore, an edge from a web service ws_1 to ws_2 is added if $op_1 \dashrightarrow op_2$ for $op_1 (\in ws_1)$ and $op_2 (\in ws_1)$—that is, partial invocation among operations.*

Consider Figure 3.1 as an example. Here, a web service network is formed by a set of web services, each of which consists of a set of operations. An operation is invoked with a set of input parameters and produces a set of output parameters. There are three kinds of nodes representing web services (e.g., ws_1 and ws_2), operations (e.g., op_{11}, op_{12} and op_{21}) or parameters (e.g., $p_1, \cdots, p_7$). Each web service node containing a set of operation nodes is connected to parameter nodes with "directed" edges. These edges show the flow of the parameters as inputs or outputs of operations. An edge from a parameter node p to an operation node op indicates that the parameter p is used as one of inputs for the operation op. On the other



```
<?xml version="1.0"?>
<definition name="WS1" ... >

<message name="WS1_OP11_Request">
<part name="P1" type="xsd:string"/>
</message>

<message name="WS1_OP11_Response">
<part name="P2" type="xsd:string"/>
<part name="P3" type="xsd:string"/>
</message>

<message name="WS1_OP12_Request">
<part name="P3" type="xsd:string"/>
<part name="P4" type="xsd:string"/>
</message>

<message name="WS1_OP12_Response">
<part name="P5" type="xsd:string"/>
<part name="P6" type="xsd:string"/>
</message>

<portType name="WS1">
<operation name="OP11">
<input message="tns:WS1_OP11_Request"/>
<output message="tns:WS1_OP11_Response"/>
</operation>

<operation name="OP12">
<input message="tns:WS1_OP12_Request"/>
<output message="tns:WS1_OP12_Response"/>
</operation>
</portType>
</definitions>
```

Figure 3.1. Web service networks: (a) WSDLs, (b) conceptual networks, (c) networks from diverse models, (d) $\mathcal{M}_p$, (e) $\mathcal{M}_{op}^f$, (f) $\mathcal{M}_{op}^p$, and (g) $\mathcal{M}_{ws}$

hand, an edge from an operation node op to a parameter node p represents that the operation op produces the parameter p as an output. For example, p_1 is one of inputs of op_{11} in ws_1. Similarly, p_5 is not only an output of op_{12} but also an input of op_{21} in ws_2.

In order to study subtle differences among node types, one can "project out" the aforementioned web service network into three kinds as follows:

- A *parameter node network*, i.e., $\mathcal{M}_p = (\mathrm{p}, D, \theta, I)$, consists of parameter nodes and edges representing operations that have an input parameter as the source node and an output parameter as the target node. For instance, if $op_1(IN_1, OUT_1)$ exists, then we create edges from each parameter $p \in IN_1$ to every output parameter $q \in OUT_1$. When approximate matching is used for intra-parameter matching, each node in $\mathcal{M}_p$ is in fact a set of similar nodes. That is, suppose there is an edge from p_1 ("password", string) to p_3 ("firstName", string) in $\mathcal{M}_p$. This indicates that one can retrieve "firstName" information by providing "password" information. Now, if an approximate matching method (e.g., edit distance) determines that a parameter p_1("password", string) matches a parameter p_2("passwd", string), then $p_2.name$ is merged with $p_1.name$ to form an edge: password, passwd $\rightarrow$ firstName. That is, the source node is a set of two nodes.

- An *operation node network*, i.e., $\mathcal{M}_{op} = (\texttt{op}, D, \theta, I)$, consists of nodes representing operations and edges representing invocations in-between. If an operation op_1 can (either partially or fully based on I) invoke an operation op_2, there is an edge $op_1 \rightarrow op_2$ in $\mathcal{M}_{op}$.
- A *web service node network*, i.e., $\mathcal{M}_{ws} = (\texttt{ws}, D, \theta, I)$, consists of web service nodes and edges representing the existence of invocable operations between web services.

Figure 3.1(d) describes a parameter node network $\mathcal{M}_p$ based on Figure 3.1(b). For instance, p_1 can be transformed to p_2 by op_{11}. For operation node example, op_{21} can be invoked with two input parameters both of which are produced by op_{12} in Figure 3.1(c). Therefore, there exists an edge $op_{12} \rightarrow op_{21}$ in both operation node networks (Figure 3.1(e) and Figure 3.1(f)). On the other hand, p_3, one of outputs for op_{11}, is used as one of inputs for op_{12}. However, op_{12} requires another input parameter p_4 not produced by op_{11}. In the partial invocation mode, therefore, an edge $op_{11} \rightarrow op_{12}$ is added. On the contrary, in the full invocation mode, no edge is added. Note that in the example of Figure 1, we obtain the same web service node network whether we use partial or full invocation mode. However, in general, different networks will be formed depending on the choice of invocation mode.

3.2.2.3 Data Set

From web services repository and Google, we have downloaded a total of 2,100 publicly available web services (as WSDL files). We refer to those data sets as **PUB**. The pre-processing of PUB data set consisted of four steps as follows:

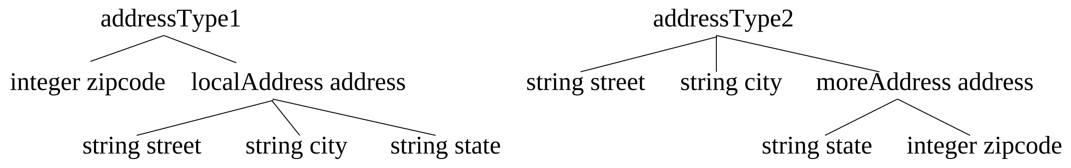


Figure 3.2. Compatible types with different structures.

(1) *Data Gathering*: First, 1,554 files were taken from Fan et al. [106] who downloaded the files from public repositories such as `XMethods.org` or `BindingPoint.com`. Second, out of top-1,000 ranked WSDL files from Google for the query string "wsdl filetype:wsdl", 546 were downloaded using Google API (the rest were undownloadable). Note that although we could have downloaded more than the first 1,000 WSDL files from Google, we decided to disregard the rest since their qualities degraded rather quickly (e.g., many WSDL files after top-1,000 were for testing as in "Hello World").

(2) *Validation & De-duplication*: According to WSDL standards, 740 invalid WSDL files were removed, and 1,360 files are left out. Then, 376 duplicate WSDL files at operation level were removed, yielding 984 valid WSDL files at the end.

(3) *Type Flattening*: In matching parameters, we use both name and type of parameters. However, since WSDL files are designed by different people in isolation, using only atomic types of XML Schema can be too rigid. For instance, consider two parameters: p_1 (address, addressType1) and p_2 (MyAddress, addressType2) of Figure 3.2. Although names of two parameters are similar, their types are user-defined and different. However, the types are in fact "compatible". Therefore, if we *flatten* these types into $p_1.type = \{\text{integer zipcode}, \text{string street}, \text{string city}, \text{stringstate}\}$ and $p_2.type = \{\text{string street}, \text{string city}, \text{string state}, \text{integer zipcode}\}$, then the parameters can be matched. We call this process type flattening[2]. After type flattening, each atomic type and name are compared using type hierarchy of XML Schema and flexible matching scheme (e.g., exact or approximate), respectively. The detailed statistics of type flattening is in Table 3.1.

(4) *Data Cleansing*: The final step is to clean data to improve the quality of parameters. For instance, a substantial number of output parameters (16%) were

[2] An alternative to our type flattening is to use more expensive metric such as tree edit distance. For its simplicity, we used type flattening in this dissertation.

Table 3.1. Type distribution

Type	Before Flattening		After Flattening	
	input (num)	output(num)	input(num)	output(num)
anyType	6.97% (7)	9.03% (5)	0.2% (35)	0.2% (51)
simpleType	65.52% (6,751)	32.37% (1,792)	92.43% (19,809)	90.63% (22,946)
string	53.75% (5,538)	22.00% (1,218)	65.06% (13,943)	54.74% (13,859)
number	7.79% (803)	7.12% (394)	17.69% (3,791)	22.89% (5,796)
time	0.90% (93)	0.22% (12)	1.85% (396)	2.93% (743)
boolean	3.74% (385)	2.46% (136)	7.16% (1,535)	9.14% (2,314)
complexType	34.41% (3,545)	67.54% (3,739)	7.41% (1,588)	9.16% (2,320)
Total Number	10,303	5,536	21,432	25,317

named “return”, “result”, or “response” which are too ambiguous for clients. However, often, their more precise underline meaning can be derived from contexts. For instance, if the output parameter named “result” belongs to the operation named “getAddress”, then the “result” is in fact “Address”. In addition, often, naming follows apparent pattern such as `getFooFromBar` or `searchFooByBar`. Therefore, to replace names of parameters or operations by more meaningful ones, we removed spam tokens like “get” or “by” as much as we could.

3.2.2.4 Distance Functions

We use three distance functions:

- *Exact Matching*: p_1 and p_2 match if $(p_1.name = p_2.name) \wedge (p_1.type = p_2.type)$.
- *Cosine Distance*: The distance of p_1 and p_2 is measured as the cosine angle of two tf/idf vectors, $v_1(p_1)$ and $v_2(p_2)$, made from the tokens of p_1 and p_2: $\cos(\theta) = 1 - \frac{v_1(p_1) \cdot v_2(p_2)}{\|v_1(p_1)\| \cdot \|v_2(p_2)\|}$.
- *WordNet-based Distance*: Since WordNet is a network of English words labeled with semantic classes (e.g., synonyms), it carries various semantic relations among words. People have proposed various ways to measure the approximate distance of two words in WordNet. We use one of them [44] that scales the information content of the least common subsumer by the

sum of the information content of two vocabularies. Formally,

$$dist_{Lin}(c_1, c_2) = \frac{2 \log p_M(c_1, c_2)}{\log p(c_1) + \log p(c_2)}$$

where $p(c)$ is the probability of encountering an instance of concept c for any concept $c \in C$, and $p_M(c_1, c_2) = \text{MIN}_{c \in S(c_1, c_2)} p(c)$ where $S(c_1, c_2)$ is the set of concepts that subsume c_1 and c_2.

Both cosine and WordNet metrics uses thresholds to determine matchmaking (i.e., if the calculated distance is below the threshold, we consider that there exists an edge in-between). Consider the following example for illustration of two matching schemes.

Example 2. *Consider two parameters p_1 = "endRoamDate" and p_2 = "EndDate" Without loss of generality, let us assume that two parameters are the same type. First, parameter names are tokenized to sets of lowercase tokens, e.g., "endRoamDate" $\rightarrow$ {"end", "roam", "date"} and "EndDate" $\rightarrow$ {"end", "date"}. Then, per each token, a tf/idf weight, w, is calculated based on the entire PUB data set as the corpora, e.g., w(date) = 0.57, w(end) = 0.57, and w(roam) = 0.7. With tf/idf weights, p_1 and p_2 are transformed to 3-dimensional vectors v_1 and v_2, respectively: $v_1 = [0.57, 0.57, 0.7]$ and $v_2 = [0.57, 0.57, 0]$. Finally, the distance between "endRoamDate" and "EndDate" is $\cos(v_1, v_2) = 1 - 0.816 = 0.184$.*

At the end, we have generated a total of 25 (= 5 network types × 5 distance metrics) web service networks: (1) three kinds of networks—$\mathcal{M}_p$, $\mathcal{M}^f_{op}$, and $\mathcal{M}^f_{ws}$, and two of their counterparts of "partial invocation"—$\mathcal{M}^p_{op}$, and $\mathcal{M}^p_{ws}$; (2) five distance variations - Exact, Cosine (0.75), Cosine (0.95), WordNet(0.75), and WordNet(0.95).

3.2.3 Main Results

We present topological landscape of web service network with respect to small world network and power-law distribution.

3.2.3.1 Small World

First, we analyze the **PUB** data set to see whether it exhibits the characteristics of small world network. Similar to previous works on the small world network [97], we restrict out attention to the giant connected component. The details of giant components of each web service network are summarized in Table 3.2. Note that the percentage of the giant component in the original network is generally bigger in the WordNet (0.75) than other cases. It indicates that the usage of the approximate matching scheme reduces the number of isolated web services in the web service network. The connection in web service network becomes important since when a web service node is not connected to the giant component, there is no way to provide web services to clients (i.e., no profit opportunity).

Table 3.3 shows the average shortest path, L, and clustering coefficient, C, for giant components of 25 web service networks, compared to random graphs with the same number of nodes and same average number of edges per node. It shows that all web service networks generated from the **PUB** data set are small world networks—$L_{actual} \gtrsim L_{random}$ and $C_{actual} \gg C_{random}$. This suggests that the real-world web services have short-cuts that connect nodes of networks. Otherwise, nodes of networks would be much farther apart than L_{random}.

There are distinct differences between matching schemes in terms of the small world properties. For example, in the parameter node network $\mathcal{M}_p$, $L_{actual}(= 3.2656)$ of the WordNet (0.75) is much smaller than $L_{actual}(= 4.3185)$ of the exact matching, while $C_{actual}(= 0.3118)$ of the WordNet (0.75) is much bigger than $C_{actual}(= 0.2229)$ of the exact matching. For better understanding, Figure 3.3 illustrates the changes of $Index_{SN}$ for different matching schemes. Recall that $Index_{SN}$ tends to increase as C increases or L decreases. In the figure, we can see that the usage of approximate matching scheme, WordNet (0.75), generates more distinct small world properties than the exact and cosine distance matching schemes. In the case of $\mathcal{M}_p$ with the WordNet (0.75) matching, $Index_{SN}$ is ten times bigger than one of the exact matching case. This result implies that the approximate matching scheme makes the network denser with the increased number of edges, so that L decreases while C increases. From the web service composition perspective, this result suggests that approximate matching scheme can facilitate more productive and effective service compositions than when only exact syntactic matching is used.

Table 3.2. Statistics of PUB: (A) # of nodes, (B) # of nodes in a giant component, (C) percentage of giant component, (D) average degree, and (ℓ) network diameter.

Scheme	Model	A	B	C (%)	D	ℓ
Exact matching	$\mathcal{M}_p$	11,301	8,494	75.1	18.66	21
	$\mathcal{M}_{op}^p$	5,180	2,993	57.7	108.01	10
	$\mathcal{M}_{op}^f$	5,180	1,538	29.6	28.25	11
	$\mathcal{M}_{ws}^p$	984	608	61.7	52.41	6
	$\mathcal{M}_{ws}^f$	984	431	43.8	17.25	7
Cosine (0.95)	$\mathcal{M}_p$	10,952	8,385	76.5	18.13	21
	$\mathcal{M}_{op}^p$	5,180	3,232	62.3	110.43	10
	$\mathcal{M}_{op}^f$	5,180	1,667	32.1	28.89	11
	$\mathcal{M}_{ws}^p$	984	656	66.6	54.42	7
	$\mathcal{M}_{ws}^f$	984	474	48.1	17.73	9
Cosine (0.75)	$\mathcal{M}_p$	10,568	8,197	77.5	17.50	9
	$\mathcal{M}_{op}^p$	5,180	3,375	65.1	110.43	9
	$\mathcal{M}_{op}^f$	5,180	1,696	32.7	28.92	11
	$\mathcal{M}_{ws}^p$	984	676	68.6	54.04	7
	$\mathcal{M}_{ws}^f$	984	485	49.2	17.69	9
WordNet (0.95)	$\mathcal{M}_p$	8,870	7,077	79.7	18.82	18
	$\mathcal{M}_{op}^p$	5,180	3,742	72.2	214.51	7
	$\mathcal{M}_{op}^f$	5,180	2,045	39.4	39.31	9
	$\mathcal{M}_{ws}^p$	984	765	77.7	86.23	6
	$\mathcal{M}_{ws}^f$	984	558	56.7	28.77	6
WordNet (0.75)	$\mathcal{M}_p$	7,042	5,590	79.3	16.87	10
	$\mathcal{M}_{op}^p$	5,180	3,967	76.5	498.01	7
	$\mathcal{M}_{op}^f$	5,180	2,574	49.6	110.58	8
	$\mathcal{M}_{ws}^p$	984	807	82.0	178.93	5
	$\mathcal{M}_{ws}^f$	984	667	67.7	66.25	5

3.2.3.2 Power-Laws

First, we examine how *complex* web services are in general. One way to measure the complexity of web services is to measure how many operations (parameters resp.) are involved in each web service (operation resp.) [106]. These results are shown in Figure 3.4, fitted to a power-law function $y = Cx^{-\alpha}$ (in log-log plot). They show *near*[3] power-law distribution with the exponent of 1.49 and 1.27, respectively. In Figure 3.4(a), 37% of the web services have just one operations and

[3]A recent study [107] reported that most of real-world power-law distributions exhibited an exponent of $2 \leq \alpha \leq 3$.

Table 3.3. Small world properties of PUB.

Scheme	Model	L_{actual}	L_{random}	C_{actual}	C_{random}
Exact matching	$\mathcal{M}_p$	4.3185	3.4244	0.2229	0.0021
	$\mathcal{M}_{op}^p$	2.8590	1.9830	0.3056	0.0362
	$\mathcal{M}_{op}^f$	3.7605	2.5628	0.2147	0.0180
	$\mathcal{M}_{ws}^p$	2.2710	1.9222	0.4809	0.0874
	$\mathcal{M}_{ws}^f$	2.9659	2.4250	0.2610	0.0405
Cosine (0.95)	$\mathcal{M}_p$	4.1760	3.4442	0.2324	0.0022
	$\mathcal{M}_{op}^p$	2.8651	1.9881	0.3125	0.0340
	$\mathcal{M}_{op}^f$	3.7538	2.5787	0.2001	0.0173
	$\mathcal{M}_{ws}^p$	2.2847	1.9254	0.4925	0.0833
	$\mathcal{M}_{ws}^f$	3.0046	2.4803	0.2499	0.0359
Cosine (0.75)	$\mathcal{M}_p$	4.1981	3.4730	0.2397	0.0020
	$\mathcal{M}_{op}^p$	2.8671	1.9925	0.3190	0.0326
	$\mathcal{M}_{op}^f$	3.7392	2.5910	0.1990	0.0172
	$\mathcal{M}_{ws}^p$	2.2923	1.9307	0.4822	0.0801
	$\mathcal{M}_{ws}^f$	2.9990	2.4394	0.2392	0.0375
WordNet (0.95)	$\mathcal{M}_p$	3.6088	3.3282	0.2612	0.0027
	$\mathcal{M}_{op}^p$	2.4234	1.9425	0.3251	0.0574
	$\mathcal{M}_{op}^f$	3.4165	2.4440	0.1493	0.0190
	$\mathcal{M}_{ws}^p$	2.1222	1.8865	0.5290	0.1138
	$\mathcal{M}_{ws}^f$	2.6215	2.1839	0.2527	0.0510
WordNet (0.75)	$\mathcal{M}_p$	3.2656	3.3665	0.3118	0.0030
	$\mathcal{M}_{op}^p$	2.0546	1.8743	0.4818	0.1256
	$\mathcal{M}_{op}^f$	2.6506	1.9657	0.1842	0.0429
	$\mathcal{M}_{ws}^p$	1.8484	1.7790	0.6697	0.2214
	$\mathcal{M}_{ws}^f$	2.2226	1.9023	0.3487	0.0993

71% of the web services have less than 5 operations. On the other hand, the largest web services have over 110 operations. Similarly, in Figure 3.4(b), 181 operations (among 5,180 operations) have less than two input/output parameters. In addition, 194 operations have no input parameter and 255 operations have no output parameter. However, the whole distribution also shows a power-law-like property. After type flattening, around 65% of operations have less than 6 parameters while the maximum number of the parameters in an operation is 360.

Second, in Figure 3.5, we examine the *popularity* of parameter names to see if some parameter names occur more often than others. X-axis is the frequency of parameter names while Y-axis is the number of samples. That is, (10, 100)

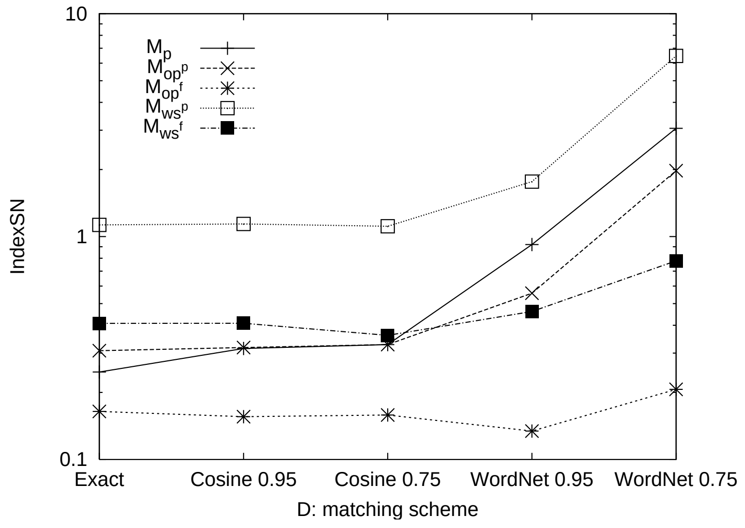
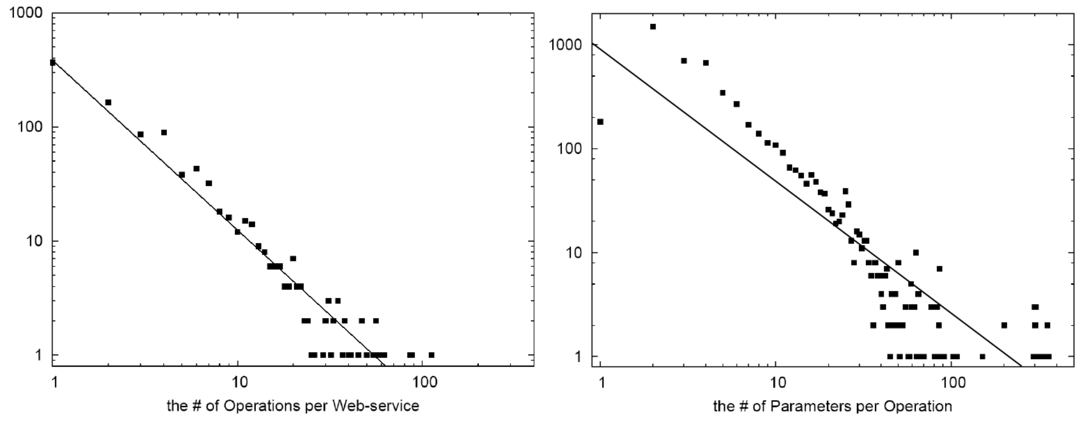


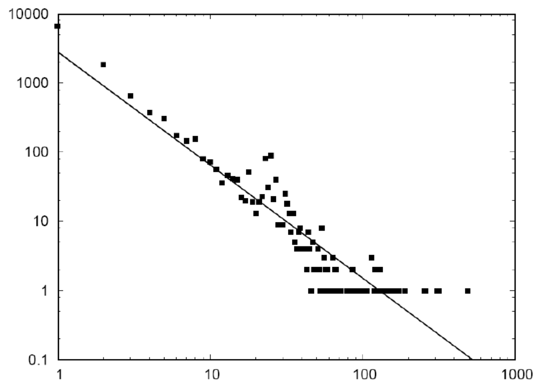
Figure 3.3. The small world index, $Index_{SN}$



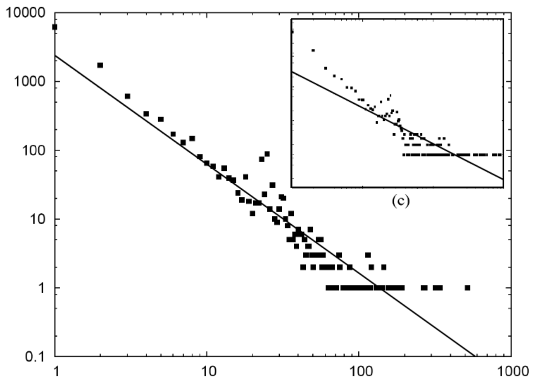
(a) α = 1.49 (b) α = 1.27

Figure 3.4. The complexity of web services and operations. Y-axis is # of samples (i.e., *ws* or *op*).

indicates that there are 100 parameters that occur 10 times. Again, all of them show near power-law distributions. From these distributions, we can observe a very small number of popular "hub" parameter names. For instance, the parameter ("name", string) appears most frequently in our collection. In addition, as the matching method gets more flexible, from (a) to (c) in Figure 3.5, more parameters are involved in matching. Since the matched parameters are considered as a single parameter, the number of distinct parameters gets smaller but the frequency of parameter names gets bigger. For example, while p_1 appears 490 times in Exact



(a) α=1.63



(b) α=1.58 & (c) α=1.06

Figure 3.5. The popularity of p names. X-axis is the frequency of p names while Y-axis is # of samples. (a) Exact matching, (b) Cosine (0.75), and (c) WordNet (0.75) (inset).

matching with 11,301 distinct parameter names, p_1 appears 3,278 times in WordNet (0.75) matching with 7,042 distinct parameter names, where similar parameters like p_1 ("name", string) and p_2 ("identification", string) are considered to be a "match" and consolidated. Therefore, Figure 3.5(c) (inset) has the smallest α and the largest tail among the three.

In Figure 3.6, we examine the out-degree distributions of three web service networks of three matching schemes. Unlike previous figures, by and large, out-degree distributions no longer follow power-law distributions well (although we still fit them with power-law functions). Nevertheless, it is evident to see the existence of hub parameters, operations, or web services with huge number of out-degrees while majority has only a few.

In the parameter node networks of $\mathcal{M}_p$ of Figure 3.6(a)-(c), due to flexible matching, the number of nodes decreases from (a) to (c): 11,301 for Exact matching, 10,568 for Cosine (0.75), and 7,042 for WordNet (0.75). However, even though Figure 3.6(c) has the smallest number of nodes of 7,042, it has the largest number of edges and biggest out-degree. For example, a parameter node including ("name", string) has 317 out-degrees (to 2.7% of nodes) in Exact matching, 306 out-degrees (to 2.9% of nodes) in Cosine (0.75), and 1,378 out-degrees (to 19.57% of nodes) in WordNet (0.75). The distributions between Figure 3.6(a)-(b) show little difference with respect to the number of nodes.

Unlike parameter node networks of $\mathcal{M}_p$, all operation node networks of $\mathcal{M}_{op}^f$ of

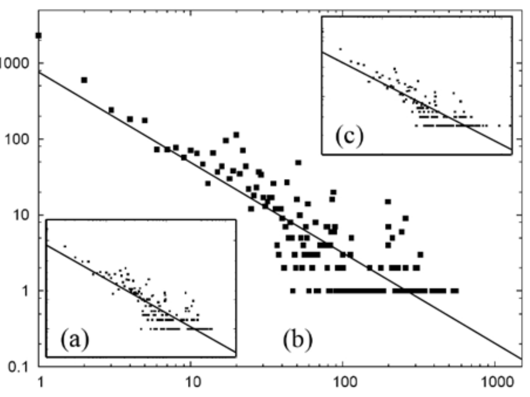
Figure 3.6(d)-(f), have 5,180 fixed operation nodes, but, from exact to approximate matching, the total number of edges increases from 22,253 to 25,174 to 184,429. The fact that approximate matching schemes such as Cosine or WordNet have more out-degrees indicates that information can flow more flexibly among web services operations through many edges. In Figure 3.6(g)-(i), the web services node networks of $\mathcal{M}_{ws}^{f}$ also have a fixed number of 984 nodes, and the out-degree increases from (g) to (i). While some edges in operation node networks include intra-connections within the same web service, the edges of $\mathcal{M}_{ws}^{f}$ are amount to pure inter-connection among different web services. Consequently, more cooperation with different web services can be expected as we have more flexible approximate matching.

3.2.4 Observations and Limitations

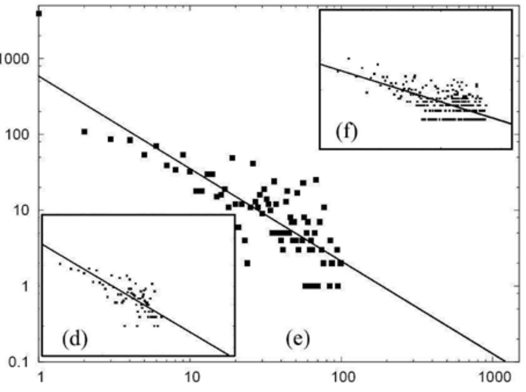
Here, we briefly discuss three important observations from (and limitations of) our experimentation.

3.2.4.1 Semantic Web

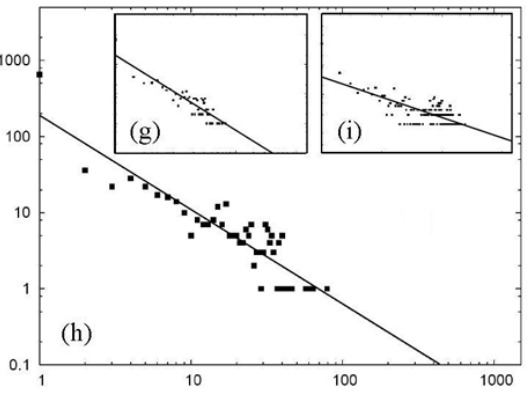
The fact that Sections 3.2.3.1 and 3.2.3.2 show the small world and power-law-like properties more clearly as the matchmaking scheme becomes more *flexible* (from Exact to Cosine to WordNet) can be interpreted as an evidence of the needs of more semantically-rich web service networks. Many real networks (that are known to be robust and error-tolerant) have shown both small world and power-law-like properties well. Therefore, we may argue that as web service networks get more mature and robust, they will show both small world and power-law-like properties better. The ultimate case of flexible matching is to use the semantics of parameters or operations. For instance, in the WordNet method of our experimentation, both "play" and "start" can be treated as synonym, meaning "begin". However, "play" can be used as a noun with the meaning "a theatrical performance" rather than a verb "begin". Therefore, this kind of mis-matching would cause a problem in discovering and composing web services (semi-)automatically. We envision that this kind of problems be solved only when the semantic web (services) are fully adopted in future.



(a)-(c) $\mathcal{M}_p$



(d)-(f) $\mathcal{M}_{op}^f$



(g)-(i) $\mathcal{M}_{ws}^f$

Figure 3.6. Out-degree distribution of three web service networks, $\mathcal{M}_p$ for (a)-(c), $\mathcal{M}_{op}^f$ for (d)-(f) and $\mathcal{M}_{ws}^f$ for (g)-(i): (a) Exact matching =1.15, (b) Cosine (0.75) =1.19, (c) WordNet (0.75) =1.04, (d) Exact matching =1.18, (e) Cosine (0.75) =1.12, (f) WordNet (0.75) =0.64, (g) Exact matching =1.25, (h) Cosine (0.75) =1.24, and (i) WordNet (0.75) =0.68.

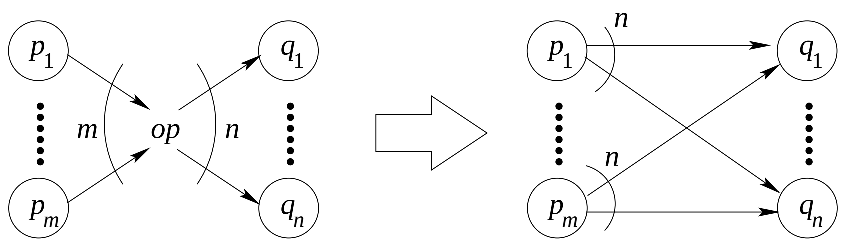


Figure 3.7. The effect of the operation complexity to the out-degree distribution.

3.2.4.2 Graph Fitting

Graphs in Figure 3.6, unlike other graphs in Section 3.2.3.2, follow a power-law-like pattern only to some extent. This is due to the existence of various outliers, which can happen for various reasons. First, the aforementioned mis-matching ("play" vs. "begin") can generate outliers. Second, the complexity of operations (e.g., the number of parameters for an operation) can significantly affect the number of outgoing edges on $\mathcal{M}_p$. For instance, in Figure 3.7, consider an operation with m input parameters and n output parameters. After transforming the operation and its parameters to $\mathcal{M}_p$, m input parameters of the operation have the same outgoing edges to all n output parameters, which generates m parameters with n out-degrees. Therefore, if m is large, then the point (n, m) would sit far above the fitting line, becoming an outlier. In $\mathcal{M}_{op}$, the complexity of web services makes similar results to $\mathcal{M}_p$. Since operations in a web service use similar parameters in names and types in general, the operations can invoke each other and have the same set of outgoing edges. As a result, operations within a web service can have the same out-degrees. Finally, we believe that outliers can happen due to the fact that we ignore service domains. Services from different domains are often not appropriate to invoke each other (i.e., ws_1 in the airline domain may not be meant to be used for ws_2 in the insurance domain), although they could have their inputs/outputs matched. We expect that as the size of web services increases and service domains are taken into account for matching, fitting becomes smoother.

3.2.4.3 Generative Model

Agreement between the web service networks and theory in Section 3.2.3.2 suggests that there be no statistical difference between a web service network and an equivalent random network in terms of the giant component size. However, at the same time, the existence of disagreeing points in the analysis also suggests that one needs better models or tools (than the simple random network model) to be able to fully explain the properties and behaviors of the web service networks. Needless to say, more investigation is needed to fully untangle this problem.

3.3 Web Service Composition using SAT Solver

In this section, we propose a novel technique to find an *optimal* composition based on techniques for the boolean satisfiability problem (SAT). Given a set of web services and a requirement web service described in WSDL, our algorithm identifies the *shortest sequence of web services* such that we can legally invoke the next web service in each step and achieve the desired requirement eventually. We first explain how this problem can be reduced into a *reachability problem* on a *state-transition system*. Then, we present our encoding to a Cnf (Conjunctive Normal Form) formula which is true if and only if there exists a path of length k from an initial state to a goal state of the state-transition system. To solve this reachability problem, we employ a state-of-the-art Sat solver, SATzilla [8]. A preliminary experiment reveals promising results where the tool finds shortest sequences with logarithmic number of invocations of a Sat solver.

3.3.1 Type-Aware Web Service Composition

In this section, we formalize the notion of web services for composition. Unlike the web service model in the previous section, where a web service is a collection of operations, now we assume that a web service own only one operation. Accordingly, a web service can be used in a place of an operation. A *web service* is a tuple $w = (I, O)$ where

- I is a finite set of *input parameters* for w.
- O is a finite set of *output parameters* for w; each input/output parameter $p \in I \cup O$ has a type t_p.

When a web service w is invoked with all the input parameters $i \in I$ with the type t_i, it returns all the output parameters $o \in O$ with the type t_o. Given two types t_1 and t_2, t_1 is a *subtype* of t_2 (denoted by $t_1 <: t_2$) if t_1 is more informative than t_2 so that t_1 can substitute for t_2 everywhere. In this case, t_2 is a *supertype* of t_1. This relation is reflexive (i.e., $t <: t$ for any type t) and transitive (i.e., if $t_1 <: t_2$ and $t_2 <: t_3$ then $t_1 <: t_3$). We assume that the type hierarchy is given; e.g. specified in OWL. Given two web services $w_1(I_1, O_1)$ and $w_2(I_2, O_2)$, we denote $w_1 \sqsupseteq_I w_2$ if w_2 requires less informative inputs than w_1; i.e., for every $i_2 \in I_2$ there exists $i_1 \in I_1$ such that $t_{i_1} <: t_{i_2}$. Given two web services $w_1(I_1, O_1)$ and $w_2(I_2, O_2)$, we denote $w_1 \sqsubseteq_O w_2$ if w_2 provides more informative outputs than w_1; i.e., for every $o_1 \in O_1$ there exists $o_2 \in O_2$ such that $t_{o_2} <: t_{o_1}$. A *Web service discovery problem* is, given a set W of available web services and a request web service w_r, to find a web service $w \in W$ such that $w_r \sqsupseteq_I w$ and $w_r \sqsubseteq_O w$.

However, it might happen that there is no single web service satisfying the requirement. In that case, we want to find a sequence $w_1 \cdots w_n$ of web services such that we can invoke the next web service in each step and achieve the desired requirement eventually. Formally, we extend the relations, $\sqsupseteq_I$ and $\sqsubseteq_O$, to a sequence of web services as follows.

- $w \sqsupseteq_I w_1 \cdots w_n$ (where $w = (I, O)$ and each $w_j = (I_j, O_j)$) if $\forall 1 \leq j \leq n$: for every $i_2 \in I_j$ there exists $i_1 \in I \cup \bigcup_{k<j} O_k$ such that $t_{i_1} <: t_{i_2}$.
- $w \sqsubseteq_O w_1 \cdots w_n$ (where $w = (I, O)$ and each $w_j = (I_j, O_j)$) if for every $o_1 \in O$ there exists $o_2 \in \bigcup_{1 \leq j \leq n} O_j$ such that $t_{o_2} <: t_{o_1}$.

Finally, given a set of available web services W and a service request w_r, a *type-aware web service composition problem* $WC = \langle W, w_r \rangle$ we focus on in this dissertation is to find a sequence $w_1 \cdots w_n$ (every $w_j \in W$) of web services such that $w_r \sqsupseteq_I w_1 \cdots w_n$ and $w_r \sqsubseteq_O w_1 \cdots w_n$. The optimal solution for this problem is to find a sequence with the minimum value for n.

3.3.2 Reduction to reachability problem

Given a type-aware web service composition problem $WC = \langle W, w_r \rangle$, the problem can be reduced into a reachability problem on a state-transition system. A *state-transition system* is a tuple $S = (X, \Sigma, T)$ where

- X is a finite set of *boolean variables*; a *state* q of S is a valuation for all the variables in X.

- Σ is a set of *input symbols*.

- $T(X, \Sigma, X')$ is a *transition predicate* over $X \cup \Sigma \cup X'$. For a set X of variables, we denote the set of primed variables of X as $X' = \{x' \mid x \in X\}$, which represents a set of variables encoding the successor states. $T(q, a, q')$ is *true* iff q' can be the next state when the input $a \in \Sigma$ is received at the state q.

Given a set $W = \{w_1, \cdots, w_n\}$ of web services where for each j, $w_j = (I_j, O_j)$, we denote as *TP* a set of types t such that there exists $p \in \bigcup(I_j \cup O_j)$ and t is the type of p. Then, we can construct a state-transition system $S = (X, \Sigma, T)$ corresponding with W as follows:

- $X = \{x_1, \cdots, x_m\}$ where $m = |TP|$; each boolean variable x_j represents whether we have an instance with the type $t_j \in T$ at a state.

- $\Sigma = W$.

- For each j, $T(q, w_j, q') = true$ where $q = (b_1, \cdots, b_m)$, $q' = (c_1, \cdots, c_m)$ (each b_k and c_k are *true* or *false*), and $w_j = (I_j, O_j)$ iff (1) for every $i \in I_j$, there exists b_k in q such that b_k is *true* and its corresponding type t_{x_k} is a subtype of the type of i (i.e., $t_{x_k} <: t_i$), (2) if b_l is *true*, c_l is also *true*, and (3) $\forall o \in O_j$: for every variable c_k in q', if its corresponding type t_{x_k} is a supertype of t_o, c_k is *true*. Intuitively, if a web service w_j is invoked at a state q where we have data instances being more informative than inputs of w_j, we proceed to a state q' where we retain all the data instances from q and acquire outputs of w_j as well as their supertypes.

In addition, from a given requirement web service $w_r = (I_{w_r}, O_{w_r})$, we encode an *initial state predicate* $Init(X)$ and a *goal state predicate* $G(X)$ as follows:

- $Init(q) = true$ where $q = (b_1, \cdots, b_m)$ iff $\forall i \in I_{w_r}$: for every variable b_j in q, if its type t_{x_j} is a supertype of t_i (i.e., $t_i <: t_{x_j}$), b_j is *true*.

- $G(q) = \mathit{true}$ where $q = (b_1, \cdots, b_m)$ iff for every output parameter $o \in O_{w_r}$, there exists b_j in q such that b_j is *true* and its type t_{x_j} is a subtype of t_o (i.e., $t_{x_j} <: t_o$).

Intuitively, we have an initial state where we possess all the data instances corresponding to the input of w_r as well as one corresponding to their supertypes. As goal states, if a state is more informative than the outputs of w_r, it is a goal state. Finally, given a type-aware web service composition problem $WC = \langle W, w_r \rangle$, we can reduce WC into a reachability problem $R = \langle S, \mathit{Init}, G \rangle$ where the shortest path from an initial state to a goal state corresponds to the shortest sequence of web services.

3.3.3 Encoding to CNF formula

Now, we study how to construct a formula $[[R]]_k$ which is true if and only if there exists a path $q_0 \cdots q_k$ of length k for a given reachability problem $R = \langle S, \mathit{Init}, G \rangle$. The formula $[[R]]_k$ is over sets $X_0, \cdots, X_k$ of variables and $W_1, \cdots, W_k$ where each X_j represents a state along the path and W_j encodes a web service invoked in each step. It essentially represents constraints on $q_0 \cdots q_k$ and $w_1 \cdots w_k$ such that $[[R]]_k$ is satisfiable if and only if q_0 is the initial state, each q_j evolves according to the transition predicate for w_j, and q_k reaches to a goal state. Formally, the formula $[[R]]_k$ is as follows:

$$[[R]]_k \equiv \mathit{Init}(X_0) \wedge \bigwedge_{0 \le j < k} T(X_j, W_{j+1}, X'_{j+1}) \wedge G(X_k)$$

Since each X_j is a finite set of boolean variables, Σ and W_j are finite, and *Init*, T and G are predicates, we can easily translate $[[R]]_k$ into a Cnf formula which is the standard input format for conventional Sat solvers.

3.3.4 Algorithm for the optimal solution

Since we can use a Sat solver with $[[R]]_k$ to check whether there exists a path of length k from the initial state to a goal state, we are able to find a shortest path simply by increasing the value k from 0 to $|W|$. In the worst case, we check the

Algorithm 1: WebServiceCompositionLinear

```
Input  : a set $W$ of web services and a web service $w_r$.
Output: a sequence of web services.
1 $(S, Init, G) := ReduceToReachabilityProblem(W, w_r)$;
2 for $(k := 0; k \leq |W|; k := k + 1)$ do
3     $f := ConstructCNF(S, Init, G, k)$;
4     if $((path := \textsc{Sat}(f)) \neq null)$ then
5         return $ExtractWSSequence(path)$;
6     end if
7 end for
```

formula until only $|W|$ as k since multiple executions of any $w \in W$ do not provide more data instances than a single execution of w.

Algorithm 1 presents the linear version. Given a set W of web services and a requirement web service w_r, the algorithm first reduces them into a state-transition system, and initial and goal predicates as Section 3.3.2 (line 1), and it begins with 1 as the value of k. For each loop, it constructs a CNF formula for k as Section 3.3.3 (line 3), and checks it with an off-the-shelf Sat solver (line 4). If the formula is satisfiable, the Sat solver returns a truth assignment; otherwise, it returns *null*. Once the algorithm finds a path of the length k, it extracts a web service sequence from the path, and returns the sequence (line 5).

However, we can improve our algorithm from this linear algorithm based on Proposition 1.

Proposition 1. *When there does not exist a path of length k from the initial state to a goal state, there does not exist a path of length $j < k$ from the initial state to a goal state.*

Proof 1. *(By contradiction) In our problem setting, if there exists such a path π of length $j < k$ to a goal, then we also have a path π' of length k to the goal by invoking any web service w after π since the invocation of w does not lose any data instance already acquired.*

Our algorithm begins with a pivot value as k. If we find a path of length k, then we again execute the Sat solver with $[[R]]_{(low+k)/2}$ to check whether there exists a shorter path. Otherwise, we retry with $[[R]]_{(k+high)/2}$ as *binary search algorithms*. Using this manner, we can quickly converge to the shortest path. Algorithm 2

Algorithm 2: WebServiceCompositionLogarithmic

```
Input  : a set $W$ of web services and a web service $w_r$.
Output: a sequence of web services.
1  $(S, Init, G) := ReduceToReachabilityProblem(W, w_r)$;
2  $low := 0$;
3  $high := |W|$;
4  $path := null$;
5  $k := pivot$;    /* $pivot$ is predefined. */
6  while ($low \le high$) do
7      $f := ConstructCNF(S, Init, G, k)$;
8      if (($tmp := \textsc{Sat}(f)) \ne null$) then
9          $high := k - 1$;
10         $path := tmp$;
11     end if
12     else  $low := k + 1$;
13     $k := (high + low)/2$;
14 end while
15 return $ExtractWSSequence(path)$;
```

presents our logarithmic algorithm to find an optimal solution by only $\log|W|$ executions of a Sat solver which is the most expensive operation in our algorithm. Our algorithm first reduces W and w_r into a state-transition system, and initial and goal predicates (line 1), and it begins with a predefined pivot value as k (line 5). We repeat lines 6–12 until our binary search is completed. In each loop, we construct a CNF formula for k, and check it with a Sat solver. Once we find a path of length k, we again search a shorter path between *low* and $k - 1$. If not, we try to find a path between $k + 1$ and *high*. Finally, after completing the loop, our algorithm extracts a web service sequence from the shortest path we have found, and returns it (line 13).

3.3.5 Preliminary Experiment

We have implemented an automatic tool for the logarithmic algorithm in Section 3.3.4. Given a type hierarchy in a OWL file, and a set of available web services and a query web service in WSDL files, our tool generates a web service sequence in WSBPEL to satisfy the request. Since public web service networks have the small world property [108], we have selected a smaller value for the pivot

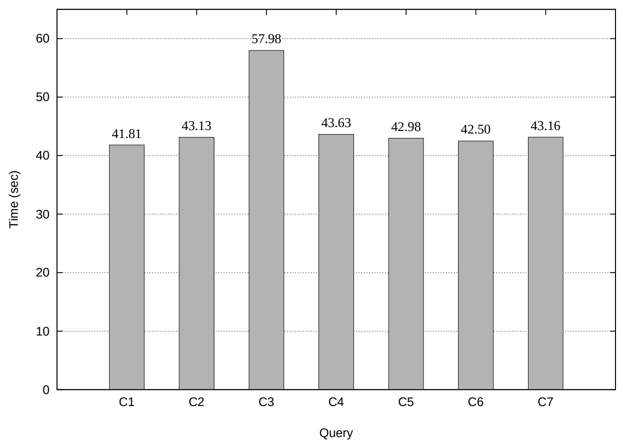


Figure 3.8. Preliminary experimental result

than the median (we use 10 as the pivot value). To demonstrate that our tools efficiently identify an optimal solution, we have experimented on the sample example for WSC'07 competition [100] which includes 413 web services and 7 queries. For an off-the-shelf Sat solver, we employ SATzilla [8] which achieved promising success in Sat 2007 competition [109]. All experiments have been performed on a PC using a 2.4GHz Pentium processor, 2GB memory and a Linux operating system. Figure 3.8 presents the total execution time[4] in seconds for our preliminary experiment of our logarithmic algorithm.

[4]We exclude the time for reduction to the state-transition system as the policy of WSC'08 does not include the bootstrap time in the total execution time.

Chapter 4

Behavior Description-based Web Service Composition

In the signature-level approach, users can simply invoke the sequence of web services computed. They cannot, however, react based on the output values returned from web services during runtime. Therefore, for better solution, we consider additional information, that is, the *behavioral descriptions* of web services.

A behavioral description of a web service is a formal specification on what the web service executes internally and externally with interacting with users; e.g., describing what output value it returns for a given input and its state, and how it changes its internal state. The behavioral description languages such as *BPEL* [11] specify service behaviors and interactions with other services as a sequence of activities. The behaviors of web services are specified as a *state-transition system*. That means, it is formally described that on receiving a specific input, what values for outputs a web service will return and what state it will proceed to. The composition with this information computes a strategy to guarantee achieving the goal can be represented as a state-transition system called a *coordinator*. Hence, given a set of behavioral descriptions of web services and a goal, web service composition problem is to synthesize a *coordinator web service* that controls the set of web services to satisfy the goal.

In this chapter, we first define a realistic model for the web service composition problem on behavioral descriptions with a state-transition system, and investigate the computational complexities for the non-deterministic web services composition problem on restricted (i.e., with full observation) and general cases (i.e., with partial observation) with concrete proof based on an alternating Turing machine (ATM). Furthermore, to solve such high complexities, we propose two approximation-based algorithms using *abstraction* and *refinement*.

4.1 Preliminaries

In this chapter, we first define a realistic model for the WSC problem with a state-transition system. For your understanding, a "Travel Agency System", one of popular example for the WSC problem, is described showing how our definition can model the (composite) service. In addition, we introduce an *Alternating Turing Machine* (*ATM*) and the *complexity classes* which will be used for the proof of the complexity in the chapter 4.2.

4.1.1 Example: Travel agency system

Consider that clients want to make reservations for both of a flight ticket and a hotel room for a particular destination and a period. However, only an airline reservation (AR) web service and a hotel reservation (HR) web service have been built separately. Clearly, we want to combine these two web services rather than implementing a new one. One way to combine them is to construct a coordinator web service which communicates with them to book up a flight ticket and a hotel room together.

Fig 4.1 illustrates this example. AR service receives a request including departing/returning dates, an origin and a destination, and then checks if the number of available seats for flights is greater than 0. If so, it returns the flight information and its price; otherwise, it returns "Not Available". Once offering the price, it waits for "Accept" or "Refuse" from its environment (in this case, a coordinator to be constructed). According to the answer, it processes the reservation. Likewise, HR service is requested with check-in/check-out dates and a location, and then checks the number of available rooms in an appropriate hotel. If there are available accommodations, it returns the room information and its price; otherwise, it returns "Not Available". AR then processes a reply "Accept" or "Refuse"

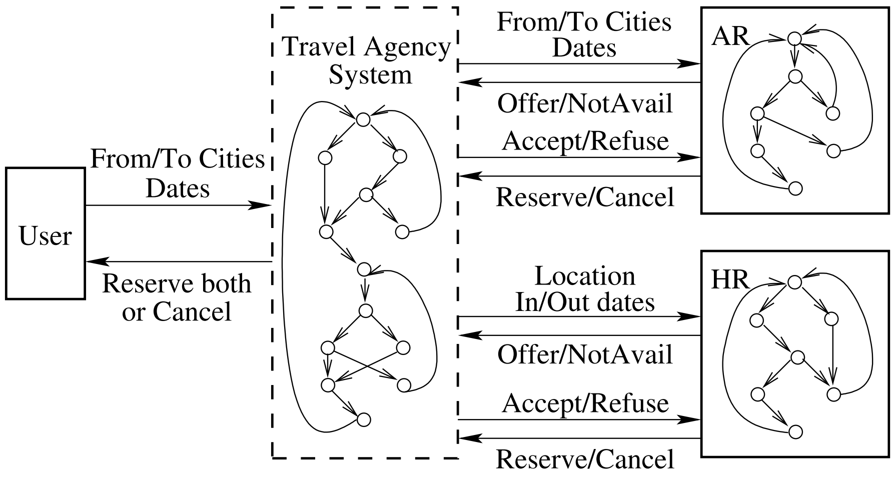


Figure 4.1. Travel agency system

from its environment.

The coordinator web service to be constructed receives from a user a request including departing/returning dates, an origin and a destination and, tries to achieve a goal, "reserve a flight ticket and a hotel room both OR cancel it", by controlling these two web services. For every output from AR and HR, the coordinator has to decide *one input* to them for the next action based on *only output values* (since in run-time it cannot access the internal variables in AR and HR, e.g. the number of available seats in flights), and it should accomplish the aim eventually. The coordinator can be represented by a deterministic state-transition system obviously.

4.1.2 Web service composition on behavioral aspect

A *web service* w is a 5-tuple $(X, X^I, X^O, Init, T)$ with the following components:

- X is a finite set of *variables* that the web service w controls; a state s of w is a valuation for every variable in X, and we denote a set of all the states as S.

- X^I is a finite set of *input variables* which w reads from its environment; $X \cap X^I = \varnothing$, and every variable $x \in X \cup X^I$ has a finite domain (e.g. Boolean, bounded integers, or enumerated types). A state *in* for inputs is a valuation for every variable in X^I, and we denote a set of all the input states as S^I.

- $X^O \subseteq X$ is a finite set of *output variables* its environment can read.
- $Init(X)$ is an *initial predicate* over X; $Init(s) = true$ iff s is an initial state. We assume that a web service has only one initial state[1].
- $T(X, X^I, X')$ is a *transition predicate* over $X \cup X^I \cup X'$. For a set X of variables, we denote the set of primed variables of X as $X' = \{x' \mid x \in X\}$, which represents a set of variables encoding the successor states. $T(s, in, s')$ is *true* iff s' can be a next state when the input $in \in S^I$ is received at the state s. T can define a *non-deterministic* transition relation.

Definition 9 (Deterministic/non-deterministic web service)**.** *A web service $w(X, X^I, X^O, Init, T)$ is deterministic if T is deterministic; namely, if for every state s and every input in, there exists only one next state s' (i.e., $T(s, in, s') = true$). Otherwise, w is non-deterministic.*

Given a state s over X (i.e. a valuation for all the variables in X) and a variable $x \in X$, $s(x)$ is the value of x in s. For a state s over X , let $s[Y]$ where $Y \subseteq X$ denote the valuation over Y obtained by restricting s to Y. For every $x \in X^I$, we add a special value *null* into its domain. If a web service receives *null* as the value for any input variable, it stays at the same state; formally, if $T(s, in, s') = true$ such that $in(x) = null$ for some $x \in X^I$, then $s = s'$.

Example 3. *Consider a simple version of a web service w for the airline reservation in Example 4.1.1, and assume that clients can request (reserve or refuse) a flight ticket by an action req_1 or req_2 (*accept *or* refuse*, respectively). The web service w can be represented as $(X, X^I, X^O, Init, T)$ where:*

- $X = \{\texttt{state}, \texttt{avail}, \texttt{reply}, \texttt{confirm}, \texttt{f\_num}, \texttt{tr\_num}\}$ *where* `state` *has the domain* $\{q_1, q_2\}$*,* `avail` *is boolean,* `reply` *has the domain* $\{undecided, offer, notAvail\}$*,* `confirm` *has the domain* $\{undecided, reserve, cancel\}$*,* `f_num` *(flight number) has the domain* $\{f_1, f_2\}$*, and* `tr_num` *(transaction number) has the domain* $\{t_1, t_2\}$*.*
- $X^I = \{\texttt{action}\}$ *where* `action` *has the domain* $\{req_1, req_2, accept, refuse\}$*.*
- $X^O = \{\texttt{reply}, \texttt{confirm}, \texttt{f\_num}\}$.

[1]We can simply simulate multiple initial states with ϵ-transitions.

- $\mathit{Init}(X) \equiv (\texttt{state} = q_1) \wedge (\texttt{reply} = \mathit{undecided})$
  $\wedge(\texttt{confirm} = \mathit{undecided})$.
- $T(X, X^I, X') \equiv$
  $(((\texttt{state} = q_1) \wedge (\texttt{action} = \mathit{req}_1) \wedge (\texttt{avail} = \mathit{true})) \rightarrow$
  $((\texttt{state}' = q_2) \wedge (\texttt{reply}' = \mathit{offer}) \wedge (\texttt{tr\_num}' = t_1)))$
  $\wedge\ (((\texttt{state} = q_1) \wedge (\texttt{action} = \mathit{req}_1)) \rightarrow (\texttt{f\_num}' = f_1))$
  $\wedge\ \cdots$
  $\wedge\ (((\texttt{state} = q_2) \wedge (\texttt{action} = \mathit{accept})) \rightarrow$
  $((\texttt{state}' = q_1) \wedge (\texttt{confirm}' = \mathit{reserve})))$
  $\wedge\ (((\texttt{state} = q_2) \wedge (\texttt{action} = \mathit{refuse})) \rightarrow$
  $((\texttt{state}' = q_1) \wedge (\texttt{confirm}' = \mathit{cancel})))$.

Note that the process model for any web service described in Semantic Web languages (e.g. OWL-S [12] or WSBPEL [11]) can be easily transformed into our representation above without any information loss if it has only finite domain variables and no recursion[2].

In the WSC problem, given a set of available web services, W, every web service in W communicates only with their coordinator but not with each other. Based on this assumption, given a set $W = \{w_1, \cdots, w_n\}$ of web services where each $w_i = (X_i, X_i^I, X_i^O, \mathit{Init}_i, T_i)$, W also can be represented by a tuple $(X, X^I, X^O, \mathit{Init}, T)$ where

- $X = X_1 \cup \cdots \cup X_n$.
- $X^I = X_1^I \cup \cdots \cup X_n^I$.
- $X^O = X_1^O \cup \cdots \cup X_n^O$.
- $\mathit{Init}(X) = \mathit{Init}_1 \wedge \cdots \wedge \mathit{Init}_n$.
- $T(X, X^I, X') = T_1 \wedge \cdots \wedge T_n$.

Since a *coordinator web service* is also a web service, it is a tuple $c = (X_c, X_c^I, X_c^O, \mathit{Init}_c, T_c)$. Although T_c can define a non-deterministic transition relation, in this problem we want only a *deterministic* transition relation for c; i.e., for every s and in, there exists only one s' such that $T_c(s, in, s') = \mathit{true}$.

[2]For this translation, see [110].

Definition 10 (Execution tree). *Given a set $W = (X,\ X^I,\ X^O,\ Init,\ T)$ of web services and a coordinator $c = (X_c,\ X_c^I,\ X_c^O,\ Init_c,\ T_c)$ where $X^I = X_c^O$ and $X^O = X_c^I$, we can define an* execution tree*, denoted by $W\|c$, which represents the composition of W and c as follows:*

- *Each node in $W\|c$ is in $S \times S_c$.*
- *The root node is (s, s_c) such that $Init(s) = true$ and $Init_c(s_c) = true$.*
- *For each node (s, s_c), it has a set of child nodes, $\{(s', s_c') \mid T(s, in, s') = true, in{=}s_c[X^I],\ T_c(s_c,\ in_c,\ s_c'){=}true, in_c{=}s'[X^O]\}$.*

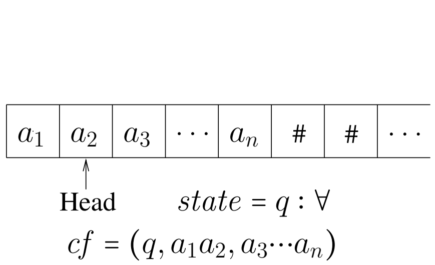
In the above, intuitively, the web services W, by receiving the input in from the current state s_c of the coordinator, collectively proceeds from s to the next state s', and then the coordinator, by receiving the input in_c from the new state s' of the web services, proceeds from s_c to the next state s_c'. Even though the composition of W and c is defined as synchronous communication, we can easily extend this model for *asynchronous* communication using τ-transition [76]. A *goal* $G \subseteq S$ is a set of states to reach, and specified as a predicate. Given a set W of web services, a coordinator c, and a goal G, we define $W\|c \models G$ if for every path $(s^0, s_c^0)(s^1, s_c^1)\cdots$ in the execution tree $W\|c$, there exists $i \geq 0$ such that $s^i \in G$; namely, every path from the initial node (s^0, s_c^0) reaches a goal state eventually.

Definition 11 (Web service composition problem). *The* web service composition (WSC) problem *that we focus on in this thesis is, given a set W of web services and a goal G, to construct a coordinator web service c such that $W\|c \models G$.*

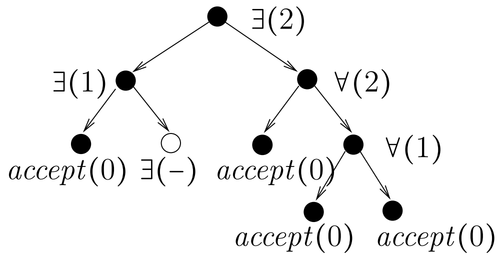
Example 4. *In Example 4.1.1, we wish to reserve both a flight ticket and a hotel room. This can be represented as $G \equiv (\texttt{flightConfirm} = reserve) \wedge (\texttt{hotelConfirm} = reserve)$. Now, given a set $W = \{w_{AR}, w_{HR}\}$ of web services and the goal G above, a WSC problem is to construct a coordinator web service c such that $W\|c \models G$.*

To study the computational complexity for web service composition problems, we define two WSC problems and the corresponding decision problems as follows:

- **WSC with complete information**: a special case of WSC problems where $W = (X, X^I, X^O, Init, T)$ such that $X = X^O$; i.e., W contains no local variable.



(a) Snapshot of ATM



(b) Computation tree

Figure 4.2. Alternating Turing machine

- **WSC with incomplete information**: a general WSC problem where there is no restriction for X^O. That is, a coordinator can read only the output variables for W.

- **Decision problem for WSC with complete information**: given $W = (X, X^I, X^O, Init, T)$ where $X^O = X$ and a goal $G \subseteq S$, is there a coordinator web service c such that $W \| c \vDash G$?

- **Decision problem for WSC with incomplete information**: given a set W of web services and a goal G, is there a coordinator c such that $W \| c \vDash G$?

4.1.3 Alternating Turing Machine

Definition 12 (Alternating Turing machine). *An alternating Turing machine (ATM) [111, 112, 113] (see Figure 4.2(a)) is a tuple $A = (Q, \Sigma, q_0, \delta, l)$ where:*

- *Q is a finite set of states.*

- *Σ is a finite set of tape alphabets.*

- *$q_0 \in Q$ is the initial state.*

- *$\delta : Q \times \Sigma \to 2^{Q \times \Sigma \cup \{\#\} \times \{\mathcal{L}, \mathcal{N}, \mathcal{R}\}}$ is a transition function where $\{\mathcal{L}, \mathcal{N}, \mathcal{R}\}$ represents the R/W head movement (i.e., it moves left, stays, and moves right).*

- *$l : Q \to \{\forall, \exists, accept\}$ is a* labeling function *for states.*

A *configuration* of an ATM $A(Q, \Sigma, q_0, \delta, l)$ is a tuple $cf = (q, \sigma, \sigma')$ where $q \in Q$ is the current state, $\sigma \in \Sigma^*$ is the tape contents left of the R/W head with rightmost symbol under the R/W head, and $\sigma' \in \Sigma^*$ is the tape contents strictly right of the R/W head. Figure 4.2(a) shows an example of the configuration, $cf = (q, a_1 a_2, a_3 \cdots a_n)$.

Given an ATM A with an input string $a\sigma$, the initial configuration is $cf_0 = (q_0, a, \sigma)$. Then, we define *successor configurations*, denoted by $\vdash$ (i.e., $(q_1, \sigma_1, \sigma'_1) \vdash (q_2, \sigma_2, \sigma'_2)$ iff $(q_2, \sigma_2, \sigma'_2)$ is a successor of $(q_1, \sigma_1, \sigma'_1)$), as follows:

- If $(q', a', \mathcal{L}) \in \delta(q, a)$, $(q, \sigma a, \sigma') \vdash (q', \sigma, a'\sigma')$.
- If $(q', a', \mathcal{N}) \in \delta(q, a)$, $(q, \sigma a, \sigma') \vdash (q', \sigma a', \sigma')$.
- If $(q', a', \mathcal{R}) \in \delta(q, a)$, $(q, \sigma a, b\sigma') \vdash (q', \sigma a' b, \sigma')$.
- If $(q', a', \mathcal{R}) \in \delta(q, a)$, $(q, \sigma a, \epsilon) \vdash (q', \sigma a' \#, \epsilon)$ where ϵ is the *empty string* and # is the *blank symbol*.

Definition 13 (Computation tree). *Given an ATM A and an input string σ, we can construct the* computation tree *(where each node in the tree is a configuration of A) as follows:*

- *The root node is the initial configuration.*
- *For each node $cf = (q_i, \sigma_i, \sigma'_i)$, a set of child nodes for cf is $\{(q_{i+1}, \sigma_{i+1}, \sigma'_{i+1}) \mid (q_i, \sigma_i, \sigma'_i) \vdash (q_{i+1}, \sigma_{i+1}, \sigma'_{i+1})\}$.*

Definition 14 (d-accepting configuration). *Given an ATM A and its input string σ, to see if A accepts σ, we define d-*accepting *for configurations in its computation tree by the bottom-up manner:*

- (q, σ, σ') *is* 0-accepting *if $l(q) = accept$.*
- (q, σ, σ') *with $l(q) = \forall$ is d-*accepting *if all successor nodes are d'-accepting for some $d' < d$ and $max(d') = d$–1.*
- (q, σ, σ') *with $l(q) = \exists$ is d-*accepting *if at least one of its child nodes is d'-accepting for some $d' < d$ and $min(d') = d$–1.*

Figure 4.2(b) illustrates a computation tree where each node (configuration) has a label (e.g., $\forall, \exists$ or *accept*) for the corresponding state and the numbers in parentheses represent that the corresponding state is some d-accepting. In this example, the node with $\exists(-)$ is not some d-accepting since there is no successor node which is some d-accepting. However, the root node (the initial configuration) is 2-accepting. Finally, A accepts σ (i.e., $\sigma \in L(A)$) if the initial configuration is d-accepting for some $n \geq 0$.

4.1.4 Complexity classes

In this thesis, we consider complexity classes [111, 112, 113] in Table 4.1. DTIME(f) is a time consumption complexity on Deterministic Turing Machines (DTMs), and DSPACE(f) is a space consumption complexity on DTMs. Similarly, we have ATIME(f) and ASPACE(f) as time and space consumption complexities on ATMs, respectively.

The class PSPACE (EXPSPACE) is the set of decision problems that can be solved by a DTM using a polynomial (exponential) amount of memory, respectively. Similarly, the class APSPACE (AEXPSPACE) is the set of decision problems that can be solved by an ATM using a polynomial (exponential) amount of memory, respectively. Chandra et al. [114] have proved the interesting connection between time complexities on DTM and space complexities on ATM:

Theorem 1. *EXP = APSPACE, and 2-EXP = AEXPSPACE [114].*

We will use Theorem 1 in out proofs. Note that the computational complexity classes in Table 4.1 have the following relationship:

$$\text{P} \subseteq \text{PSPACE} \subseteq \text{EXP (=APSPACE)} \subseteq \text{EXPSPACE} \subseteq \text{2-EXP (=AEXPSPACE)}$$

4.2 The Computational Complexities

In this section, we study the computational complexities (lower bounds) for the behavior description-based WSC problem defined in Sec 4.1.2. Our proofs use

Table 4.1. Computational complexity

Time		
P	=	$\bigcup_{k\geq 0}$ DTIME(n^k)
EXP	=	$\bigcup_{k\geq 0}$ DTIME(2^{n^k})
2-EXP	=	$\bigcup_{k\geq 0}$ DTIME($2^{2^{n^k}}$)

Space		
PSPACE	=	$\bigcup_{k\geq 0}$ DSPACE(n^k)
APSPACE	=	$\bigcup_{k\geq 0}$ ASPACE(n^k)
EXPSPACE	=	$\bigcup_{k\geq 0}$ DSPACE(2^{n^k})
AEXPSPACE	=	$\bigcup_{k\geq 0}$ ASPACE(2^{n^k})

reductions from ATMs that we defined in Sec 4.1.3, which show the space complexities for the problems. Finally, based on Theorem 1, we establish the time complexities.

4.2.1 The Complexity of WSC with complete information

Theorem 2. *The WSC problem with complete information is EXP-hard.*

The proof is to simulate an ATM with a polynomial tape length. That is, for any ATM A and an input string σ, we can construct a WSC problem in polynomial time such that A accepts σ if and only if there exists a coordinator to satisfy a goal. We prove it using the following lemmas.

Lemma 1. *Given an ATM $A = (Q, \Sigma, q_0, \delta, l)$ with polynomial space bound $p(n)$ and an input string $\sigma = a_1 \cdots a_n$ (where $n = |\sigma|$), we can construct a WSC problem instance with W and a goal G.*

Proof 2. *We can construct a set $W(X, X^I, X^O, Init, T)$ of web services and a goal G which have a polynomial size in the size of the description of A and σ as follows. The set X of variables includes the following variables:*

- *state represents the current state of A; so, it has the domain, $\{q \mid q \in Q\}$.*
- *For $1 \leq i \leq p(n)$, cl_i has the contents of the ith tape cell; its domain is $\Sigma \cup \{\#\}$.*
- *hd describes the R/W head position; its domain is $\{1, \cdots, p(n)+1\}$.*
- *label represents the label of the current state; it has the domain, $\{\forall, \exists, accept\}$.*

The set of input variables is $X^I = \{input\}$ *where the domain of input is* $\{A_{(q,i,a)} \mid q{\in}Q, l(q){=}\forall, 0{\le}i{\le}p(n), a{\in}\Sigma\} \cup \{E_{(q,i,a,j)} \mid q{\in}Q, l(q){=}\exists, 0{\le}i{\le}p(n), a{\in}\Sigma, 0{\le}j{\le}|\delta(a,q)|\}$. *The set* X^O *of output variables equals to* X *since this problem is the complete information problem. As the initial configuration of* A*, the initial state predicate* $Init(X)$ *is* $(state{=}q_0) \wedge \bigwedge_{1\le i\le n}(cl_i{=}a_i) \wedge \bigwedge_{n<i\le p(n)}(cl_i{=}\#) \wedge (hd{=}1) \wedge (label{=}l(q_0))$. *Note that the input string is* $\sigma = a_1 \cdots a_n$. *The transition predicate* $T(X, X^I, X')$ *is* $((hd{=}p(n){+}1) \to T_V) \wedge ((label{=}\forall) \to T_\forall) \wedge ((label{=}\exists) \to T_\exists)$ *with the following sub-formulae*

- $T_V \equiv (state'{=}state) \wedge (hd'{=}hd) \wedge (label'{=}label)$
- $T_\forall \equiv \bigwedge_{q\in Q, 1\le i\le p(n), a\in\Sigma}(((state = q) \wedge (hd = i) \wedge (cl_i = a) \wedge (input = A_{(q,i,a)})) \to \bigvee_{1\le j\le k}((state'{=}q_j) \wedge (cl'_i{=}a_j) \wedge \bigwedge_{m\ne i}(cl'_m{=}cl_m) \wedge (hd'{=}hd + \Delta) \wedge (label'{=}l(q_j))))$
- $T_\exists \equiv \bigwedge_{q\in Q, 1\le i\le p(n), a\in\Sigma, 1\le j\le k}(((state{=}q) \wedge (hd{=}i) \wedge (cl_i{=}a) \wedge (input{=}E_{(q,i,a,j)})) \to ((state'{=}q_j) \wedge (cl'_i{=}a_j) \wedge \bigwedge_{m\ne i}(cl'_m{=}cl_m) \wedge (hd'{=}hd + \Delta) \wedge (label'{=}l(q_j))))$

where (q_j, a_j, m_j) *is obtained from* $\delta(q,a) = \{(q_1, a_1, m_1), \cdots, (q_k, a_k, m_k)\}$*, and* $\Delta{=}-1$ *if* $m_j{=}L$*,* $\Delta{=}0$ *if* $m_j{=}N$ *and* $\Delta{=}1$ *if* $m_j{=}R$*. Note that the value for the variable input is provided by a coordinator* c*. Finally, we have a goal,* $G = \{s{\in}S \mid s(label) = accept\}$*.*

If the ATM A *violates the space bound, the variable hd has the value* $p(n){+}1$*, and after this point we cannot reach goal states since* W *stays the same state and the same head position forever by* T_V*.*

Lemma 2. *If* $\sigma \in L(A)$*, then there exists a coordinator* $c = (X_c, X^I_c, X^O_c, Init_c, T_c)$ *such that* $W \| c \models G$*.*

Proof 3. *As Sec 12,* $\sigma \in L(A)$ *means that the initial configuration of* A *with respect to* σ *is* m*-accepting* ($m \ge 0$)*. Now, we show that there exists a coordinator* c *such that for every path* $(s^0, s^0_c)(s^1, s^1_c)\cdots$ *in the execution tree* $W\|c$*, there exists* $s^i \in G$*. The coordinator to be constructed is* $c = (X_c, X^I_c, X^O_c, Init_c, T_c)$ *where* $X_c = \{input\}$*,* $X^I_c = X$*, and* $X^O_c = \{input\}$*. We will provide* $Init_c$ *and* T_c *in the later of this proof.*

When A *accepts* σ*, we can define an accepting computation tree* $ACT_{(A,\sigma)}$ *of* A *with respect to* σ *from its computation tree* Υ *as follows:*

- *For each configuration $cf = (q, \sigma_1, \sigma_2) \in \Upsilon$ such that $l(q) = \forall$, all the successor configuration are also included in $ACT_{(A,\sigma)}$. Note that if cf is m-accepting, each successor is at most $(m\text{–}1)$-accepting.*

- *For each $cf = (q, \sigma_1, \sigma_2) \in \Upsilon$ such that $l(q) = \exists$ and cf is m-accepting, only one successor configuration cf' which is $(m\text{–}1)$-accepting is included in $ACT_{(A,\sigma)}$.*

When A and σ are clear from the context, we drop the subscript $_{(A,\sigma)}$ and write ACT. Let $\sigma[i]$ be the i-th symbol of the string σ. Then, $ACT_{(A,\sigma)}$ is mapped into an execution tree $W\|c$. For this mapping, we have two mapping functions, α and β; α maps a configuration cf in ACT to a state s of web services W, and β maps cf to a state s_c of the coordinator c. First, for each $cf = (q, \sigma_1, \sigma_2)$, we have a corresponding state $s = \alpha(cf)$ of W such that

- *$s(state) = q$.*

- *For $1 \le i \le |\sigma|$, $s(cl_i) = \sigma[i]$ where $\sigma = \sigma_1\sigma_2$, and for $|\sigma| < i \le p(n)$, $s(cl_i) = \#$.*

- *$s(hd) = |\sigma_1|$.*

- *$s(label) = l(q)$.*

Next, for each configuration $cf = (q, \sigma_1, \sigma_2)$, we have a corresponding state $s_c = \beta(cf)$ of c such that

- *If $l(q) = \forall$, then $s_c(input) = A_{(q,i,a)}$ where $i = |\sigma_1|$ and $a = \sigma_1[i]$.*

- *In the case of $l(q) = \exists$, let cf' be the only successor of cf in ACT, which is obtained by a transition (q_j, a_j, m_j) among $\delta(q, a) = \{(q_1, a_1, m_1), \cdots, (q_k, a_k, m_k)\}$ where $a = \sigma_1[|\sigma_1|]$. Now, if $l(q) = \exists$, $s_c(input) = E_{(q,i,a,j)}$ where $i = |\sigma_1|$ and $a = \sigma_1[i]$.*

According to α and β, we have an execution tree of $W\|c$ where each node is $(\alpha(cf), \beta(cf))$. Now, we show by induction, that if cf in ACT is m-accepting, every path from the corresponding node $(\alpha(cf), \beta(cf))$ in $W\|c$ reaches a goal state eventually. During the induction, we also provide the transition predicate T_c of c.
***Base case**: If $cf = (q, \sigma_1, \sigma_2)$ is 0-accepting, $s = \alpha(cf)$ is a goal state since $s(label) = l(q) = accept$.*

***Induction hypothesis**: If cf is $(m{-}1)$-accepting, we assume that every path from the corresponding node $(\alpha(cf), \beta(cf))$ in $W\|c$ reaches a goal state eventually.*

***Induction step**: There are two cases: (1) $cf = (q, \sigma_1, \sigma_2)$ is a $\forall$-configuration (i.e. $l(q) = \forall$) and (2) $cf = (q, \sigma_1, \sigma_2)$ is a $\exists$-configuration (i.e. $l(q) = \exists$).*

(1) First, for $\forall$-configurations we can define the transition T_c for the coordinator as $\bigwedge_{q\in Q, 1\le i\le p(n), a\in\Sigma}(((state{=}q) \wedge (label{=}\forall) \wedge (hd{=}i) \wedge (cl_i{=}a)) \to (input'{=}A_{(q,i,a)}))$. Note that T_c decides, based on the value for variables in X_c^I, the value for input as the input to W. Let $cf = (q, \sigma_1, \sigma_2)$ be a m-accepting $\forall$-configuration with successor configurations $cf_1, \cdots, cf_k$. For cf, we have a corresponding node (s, s_c) in $W\|c$ where $s = \alpha(cf)$ and $s_c = \beta(cf)$. Then, according to T and T_c, we have successor nodes $(s_1, s_{c1}), \cdots, (s_k, s_{ck})$. Since our transition predicates T and T_c strictly follow the transition function δ of A, these nodes are consistent with $(\alpha(cf_1), \beta(cf_1)), \cdots, (\alpha(cf_k), \beta(cf_k))$ where each cf_i is m'-accepting ($m'<m$). Consequently, by the induction hypothesis every path from each (s_i, s_{ci}) reaches a goal state. Finally, every path from $(\alpha(cf), \beta(cf))$ also reaches a goal state.

(2) Let $cf = (q, \sigma_1, \sigma_2)$ be a m-accepting $\exists$-configuration with the successor configuration cf'. Note that by the definition of ACT, m-accepting $\exists$-configurations (for any m) have only one successor configuration which is $(m{-}1)$-accepting. Let (q_j, a_j, m_j) be the transition from cf to cf' which is j-th among $\delta(q, a) = \{(q_1, a_1, m_1), \cdots, (q_k, a_k, m_k)$ where $a = \sigma_1[|\sigma_1|]$. Now, we can define, for $\exists$-configurations, the transition T_c for the coordinator as $\bigwedge_{q\in Q, 1\le i\le p(n), a\in\Sigma}(((state{=}q) \wedge (label{=}\exists) \wedge (hd{=}i) \wedge (cl_i{=}a)) \to (input'{=}E_{(q,i,a,j)}))$. For cf, we have a corresponding node (s, s_c) in $W\|c$ where $s = \alpha(cf)$ and $s_c = \beta(cf)$. Then, according to T and T_c, we have one successor node (s', s'_c). Again, since our transition predicates T and T_c strictly follow the transition function δ of A, this node is consistent with $(\alpha(cf'), \beta(cf'))$ where each cf' is $(m{-}1)$-accepting. Consequently, by the induction hypothesis, every path from (s', s'_c) reaches a goal state. Finally, every path from $(\alpha(cf), \beta(cf))$ also reaches a goal state.

Similarly with T_c, we can define the initial predicate $Init_c$ for c as $((l(q_0){=}\forall) \to (input{=}A_{(q_0,1,a_1)})) \wedge ((l(q_0){=}\exists) \to (input{=}E_{(q_0,1,a_1,j)}))$ where a_1 is the first symbol of the input string and j is the index of the transition by which the ATM proceeds from the initial configuration to the next in the case of $\exists$-initial configuration. Finally, since the initial configuration is m-accepting, every path from the initial node of

$W \| c$ reaches a goal state; i.e., $W \| c \vDash G$.

Lemma 3. *If there exists a coordinator c such that $W \| c \vDash G$, then $\sigma \in L(A)$.*

Proof 4. *As Sec 4.1.2, the fact that there exists a coordinator c such that $W \| c \vDash G$ means that every path $(s^0, s_c^0)(s^1, s_c^1)\cdots$ from the initial node in the execution tree $W \| c$ reaches a goal state eventually. Now, we show that an ACT for A corresponding to $W \| c$ can be constructed and the initial configuration is m-accepting.*

We denote as ST, a finite subtree of $W \| c$ which includes, for every path $(s^0, s_c^0)(s^1, s_c^1)\cdots$ of $W \| c$, its prefix ending at a goal state (i.e. $(s^0, s_c^0)\cdots(s^k, s_c^k)$ such that $s^k \in G$). In what follows, we construct an ACT for the ATM A from the subtree ST. For the mapping, we have a mapping function γ which maps a state s of web services W to a configuration cf of A. For each state s such that $s(state)$=q, $s(cl_i)$=b_i where $1 \le i \le p(n)$, $s(hd)$=i and $s(label)$=$l(q)$, we have a corresponding configuration $cf = \gamma(s) = (q, \sigma_1, \sigma_2)$ such that $\sigma_1 = b_1 \cdots b_i$ and $\sigma_2 = b_{i+1} \cdots b_{k-1}$ where k is the index of the first appearance of #.

Now, we claim that if among every path from a node (s, s_c) to a goal in ST, the length of the longest one is m, the corresponding configuration $\gamma(s)$ is m-accepting. We show by induction that our claim is correct.

***Base case**: If s is a goal state, $cf = \gamma(s)$ is 0-accepting since $s(label) = l(q) =$ accept.*

***Induction hypothesis**: We assume that if among every path from a node (s, s_c) to a goal in ST, the length of the longest one is m–1, the configuration $\gamma(s)$ is $(m$–$1)$-accepting.*

***Induction step**: There have two cases: (s, s_c) with (1) $s(label) = \forall$ and (2) $s(label) = \exists$.*

(1) Among every path from a node (s, s_c) to a goal in ST such that $s(label) = \forall$, let the length of the longest one be m. Let (s, s_c) have successor nodes $(s_1, s_{c1}), \cdots$, (s_k, s_{ck}) in ST. For (s, s_c), we have a corresponding configuration $cf = \gamma(s)$. Then, according to δ, cf has $cf_1, \cdots, cf_k$ as its successor configurations. Since T and T_c strictly follow the transition function δ of A, these configurations are consistent with $\gamma(s_1), \cdots, \gamma(s_k)$. In the successors of (s, s_c), some (s_i, s_{ci}) has m–1 as the longest path length and the rest have less than m–1. Thus, by the induction hypothesis the configuration $\gamma(s_i)$ for s_i is $(m$–$1)$-accepting, and the rest are less than $(m$–$1)$-accepting. Finally, the configuration $\gamma(s)$ is m-accepting.

(2) Among every path from a node (s, s_c) to a goal in ST such that $s(label) = \exists$, let the length of the longest one be m. According to our transition predicates T and T_c, (s, s_c) has only one successor (s', s'_c) which has m–1 as the length of the longest path to a goal. For (s, s_c), we have a corresponding configuration $cf = \gamma(s)$. Then, according to δ, cf has $cf_1, \cdots, cf_k$ as its successor configurations, one of which, say cf_i, is consistent with $\gamma(s')$. By the induction hypothesis the corresponding configuration $cf_i = \gamma(s')$ is $(m-1)$-accepting. Hence, the configuration $\gamma(s)$ is m-accepting.

Finally, since the initial node (s, s_c) of ST has m (for some $m \geq 0$) as the length of the longest path to a goal, the corresponding configuration $\gamma(s)$, namely the initial configuration of A, is m-accepting.

4.2.2 The Complexity of WSC with incomplete information

Theorem 3. *The WSC problem with incomplete information is 2-EXP-hard.*

The proof is to simulate an ATM with exponential tape length. As Theorem 2, we prove it by the following lemmas.

Lemma 4. *Given an ATM $A = (Q, \Sigma, q_0, \delta, l)$ with exponential space bound $e(n)$ and an input string $\sigma = a_1 \cdots a_n$ (where $n = |\sigma|$), we can construct a WSC problem instance with W and a goal G.*

Proof 5. *An important difference with the complete information problem is that we are not allowed to have a variable for each tape cell since the number of tape cells is exponential and the reduction could not be polynomial. Instead of including an exponential number of variables cl_i, we have one variable cl and its index idx. The trick is to establish that if the index matches the current head position, W should simulate the ATM A, and to force the above to be satisfied universally for every index idx. Given an ATM A with σ, we construct a set $W(X, X^I, X^O, Init, T)$ of web services and a goal G as follows. The set X of variables includes the following variables:*

- *state; its domain is $\{q \mid q \in Q\}$.*

- *idx; its domain is* $\{1, \cdots, e(n)\}$.
- *cl represents the contents of the cell of which index is idx; its domain is* $\Sigma \cup \{\#\}$.
- *hd; its domain is* $\{1, \cdots, e(n)+1\}$. *For idx and hd, we need only* $\lceil log_2(e(n)+1) \rceil$ *bits.*
- *label; it has a domain,* $\{\forall, \exists, accept\}$.
- *lsb represents the symbol written by the head in the last step; it has a domain,* $\Sigma \cup \{\#\}$.

The set X^I *is* $\{input\}$ *where the domain of input is* $\{A_{(q,a)} \mid q \in Q, l(q) = \forall, a \in \Sigma\} \cup \{E_{(q,a,j)} \mid q \in Q, l(q) = \exists, a \in \Sigma, 0 \leq j \leq |\delta(a,q)|\}$. *The set* X^O *is* $\{state, cl\}$. *Init*(X) *is* $(state{=}q_0) \wedge ((idx \leq |\sigma|) \Leftrightarrow (cl{=}a_{idx})) \wedge ((idx > |\sigma|) \Leftrightarrow (cl{=}\#)) \wedge (hd{=}1) \wedge (label = l(q_0))$. *The initial predicate allows any value for idx, and the value for cl is determined on idx. The transition predicate* $T(X, X^I, X')$ *is* $((hd{=}e(n)+1) \rightarrow T_V) \wedge ((label{=}\forall) \rightarrow T_\forall) \wedge ((label{=}\exists) \rightarrow T_\exists)$ *with the following sub-formulae*

- $T_V \equiv (state'{=}state) \wedge (hd'{=}hd) \wedge (label'{=}label)$
- $T_\forall \equiv \bigwedge_{q \in Q, a \in \Sigma}(((state = q) \wedge ((hd = idx) \rightarrow (cl = a)) \wedge (input = A_{(q,a)})) \rightarrow \bigvee_{1 \leq j \leq k}((hd{=}idx) \rightarrow ((state'{=}q_j) \wedge (cl'{=}a_j) \wedge (hd'{=}hd + \Delta) \wedge (label'{=}l(q_j)) \wedge (lsb'{=} a_j))))$
- $T_\exists \equiv \bigwedge_{q \in Q, a \in \Sigma, 1 \leq j \leq k}(((state{=}q) \wedge ((hd{=}idx) \rightarrow (cl{=}a)) \wedge (input{=}E_{(a,q,j)})) \rightarrow (((hd{=}idx) \rightarrow (state'{=}q_j) \wedge (cl'{=}a_j) \wedge (hd'{=}hd{+}\Delta) \wedge (label'{=}l(q_j)) \wedge (lsb'{=}a_j))))$

where (q_j, a_j, m_j) *is obtained from* $\delta(q,a) = \{(q_1, a_1, m_1), \cdots, (q_k, a_k, m_k)\}$ *and* $\Delta{=}{-}1$ *if* $m_j{=}L$, $\Delta{=}0$ *if* $m_j{=}N$ *and* $\Delta{=}1$ *if* $m_j{=}R$. *Finally, we have a goal,* $G = \{s \in S \mid s(label) = accept\}$.

If the ATM A violates the space bound, the variable hd has the value $e(n) + 1$, *and after this point we cannot reach goal states since W stays the same state and the same head position forever by* T_V.

Lemma 5. *If* $\sigma \in L(A)$, *then there exists a coordinator c such that* $W \| c \vDash G$.

Proof 6. *Given A such that $\sigma \in L(A)$, we can construct a coordinator $c = (X_c, X_c^I, X_c^O, Init_c, T_c)$ where $X_c = \{input\}$, $X_c^I = \{state, lsb\}$, and $X_c^O = \{input\}$. As the proof of Lemma 2, we can define T_c with a conjunction of two cases: $\forall$-state and $\exists$-state. That is, if $l(q) = \forall$, the transition predicate is $\bigwedge_{q\in Q, a\in\Sigma}(((state{=}q) \wedge (cl{=}a)) \rightarrow (input'{=}A_{(q,a)}))$. Otherwise, $\bigwedge_{q\in Q, a\in\Sigma}(((state{=}q) \wedge (cl{=}a)) \rightarrow (input'{=}E_{(q,a,j)}))$ where j is the index of the transition by which the ATM proceeds from the corresponding $\exists$-configuration to the next in $ACT_{(A,\sigma)}$. Similarly with T_c, we can define the initial predicate $Init_c$ as $((l(q_0){=}\forall) \rightarrow (input{=}A_{(q_0,a_1)})) \wedge ((l(q_0){=}\exists) \rightarrow (input{=}E_{(q_0,a_1,j)}))$ where a_1 is the first symbol of the input string σ and j is obtained as the above.*

Now, we show that $ACT_{(A,\sigma)}$ is mapped into an execution tree $W\|c$. For this mapping, we have two mapping functions, α and β; α maps a configuration cf in ACT to a state s of web services W, and β maps cf to a state s_c of the coordinator c. First, given a configuration $cf = (q, \sigma_1, \sigma_2)$ and a tape index $1{\le}i{\le}e(n)$, we have a corresponding state $s = \alpha(cf, i)$ of W such that

- $s(state) = q$.
- $s(cl) = \sigma[i]$ *if* $i \le \sigma$ *where* $\sigma = \sigma_1\sigma_2$; *otherwise,* $s(cl) = \#$.
- $s(idx) = i$.
- $s(hd) = |\sigma_1|$.
- $s(label) = l(q)$.

The mapping function β is the same with β in Lemma 2. Now, we claim that if cf in ACT is m-accepting, then for every $1 \le i \le e(n)$ every path from the corresponding node $(\alpha(cf, i), \beta(cf))$ reaches a goal state eventually. By using the property that T and T_c strictly follow the transition function δ of A, we can prove the claim by induction as Lemma 2. We omit the induction for the space limit.

Finally, since the initial configuration of ACT is m-accepting, every path from the initial node of $W\|c$ reaches a goal state; that is, $W\|c \models G$.

Lemma 6. *If there exists a coordinator c such that $W\|c \models G$, then $\sigma \in L(A)$.*

Proof 7. *For the finite subtree ST of $W \| c$, we construct $ACT_{(A,\sigma)}$. However, unlike Lemma 3, we are not able to construct a configuration directly from a state of W since W does not have all the tape contents, but only cl and lsb. Now, our trick is to construct the computation tree by a top-down manner. Even though the initial state of W has only cl and lsb, we can construct the initial configuration as $cf = (q_0, a, \sigma')$ where the input string $\sigma = a\sigma'$. Given a predecessor configuration $cf_1 = (q_1, \sigma_1, \sigma_1')$ and a state s of W such that $s(state){=}q$, $s(cl){=}a_1$, $s(idx){=}i$, $s(hd) = h$, $s(label) = l(q)$, and $s(lsb) = a_2$, our mapping function γ maps s to a configuration $cf_2 = (q, \sigma_2, \sigma_2')$ where $|\sigma_2| = h$ and for σ_2 and σ_2', $\sigma_2\sigma_2'$ is copied from $\sigma_1\sigma_1'$ except $(\sigma_2\sigma_2')[|\sigma_1|] = a_2$.*

Now, we claim that if among every path from a node (s, s_c) to a goal in ST, the length of the longest one is m, the corresponding configuration $\gamma(s)$ is m-accepting. By using the property that our T and T_c strictly follow the transition function δ of A, we can prove the claim by induction as Lemma 3.We omit the induction.

Finally, since the initial node (s, s_c) of ST has m (for some $m \geq 0$) as the length of the longest path to a goal, the corresponding configuration $\gamma(s)$, namely the initial configuration of A, is m-accepting.

4.3 Approximation Algorithms

Our findings in the previous section 4.2 suggest that much more efforts to find "alternative" solutions to the WSC problem such as an approximation algorithm be needed. Toward this challenge, we propose two approximation-based algorithms using *abstraction* and *refinement*. The first step is to reduce the original web services to the abstract ones with less variables and then try to solve the abstract problem. If we identify a coordinator that controls the abstract web services to satisfy a given goal, the coordinator can control the original web services to satisfy the goal since the abstract web services over-approximate the concrete ones. Otherwise, we refine the abstract web services by adding variables, and repeat to find a solution. For abstraction, we propose two methods—*signature-preserving abstraction* and *signature-subsuming abstraction*. Our experiment on 3 sets of realistic problems (8 instances) shows that our technique outperforms a basic algorithm without abstraction/refinement.

4.3.1 Basic Algorithm for WSC Problem

In this section, we study a basic algorithm for the general WSC problem defined in Section 4.1. Several researches [115, 76] have successfully applied a planning technique with partial observation [116] to WSC problems. Thus, we also employ the same method for our baseline algorithm; Algorithm 3 for the WSC problem is based on the automated planning algorithm on partial observation [116]. In a general case of WSC, a coordinator web service is not able to identify the exact state of target web services. Hence, we model this uncertainty by using a *belief state*, which is a set of *possible* states of target web services but *indistinguishable*. The underlying idea of Algorithm 3 is to construct an *and-or searching tree* from initial belief states to goal belief states. That is, from any node (a belief state) of the tree, for non-determinism of output values of web services, we extend the tree with a set of child nodes via *and-edges*. In this case, all the child nodes should reach a goal belief state. For coordinator's selecting input values, we construct a set of child nodes via *or-edges*. In this case, at least one child is required to reach a goal belief state.

To initialize the and-or searching tree, Algorithm 3 first constructs a root node (a belief state) corresponding to the given initial predicate, *Init*, and assigns "*undecided*" to the result value for the root (lines 1–2). If the states corresponding to *Init* are already included in goal states, we assign "*true*" to the result value for the root. Next (lines 5–12), until determining the result value for the root, we repeat: (1) to select a node which is not determined yet as "*true*" or "*false*", (2) to extend the tree from the selected node by computing a set of possible successor nodes, and (3) to check if the node can reach a goal state based on the and-or constraint. Once we identify the result of each node, we propagate the result to its ancestor node. Finally, if the algorithm identifies the result of root node as *true*, it constructs a coordinator web service from the tree, and returns the coordinator. Otherwise, it returns *null*. The complexity of the algorithm is $O(2^{2^n})$ where n is the number of variables in W, since the number of states of W is 2^n and thus the number of belief states is 2^{2^n} (recall Theorem 3).

Algorithm 3: WSC with partial observation

```
Input  : A set $W$ of web services and a goal $G$.
Output: A coordinate web service $c$.
tree := InitializeSearchingTree(Init);
tree.root.result := undecided;
if (States(Init) ⊆ States(G)) then
    tree.root.result := true;
end if
while (tree.root.result = undecided) do
    node := SelectNode(tree);
    childNodes := ExtendTree(tree, node);
    if (CheckSuccess(childNodes)) then
        node.result := true;
    end if
    else if (CheckFailure(childNodes)) then
        node.result := false;
    end if
    PropagateResult(tree, node);
end while
if (tree.root.result = true) then
    return ConstructCoordinator(tree);
end if
else return null;
```

4.3.2 Signature-preserving Abstraction and Refinement

Theorems 2 and 3 imply that the WSC problem is computationally hard. Hence, more efforts to devise efficient approximation solutions to the WSC problem are needed. In addition, the complexity of Algorithm 3 also provides the same implication. Therefore, we propose two approximation-based methods using abstraction and refinement in Sections 4.3.2 and 4.3.3.

4.3.2.1 Signature-preserving abstraction

Given a set W of web services, we define *signature-preserving abstract web services* that have the same signature (i.e., the same I/O variables) but less variables than W.

Definition 15 (Signature-preserving abstract web services)**.** *Given a set of web services $W(X, X^I, X^O, Init, T)$ and a set Y of variables such that $X^{IO} \subseteq Y \subseteq X^A$,* *the signature-preserving abstraction of W with respect to Y is $W_Y(X_Y,\ X_Y^I,\ X_Y^O,\ Init_Y,\ T_Y)$ where:*

- $X_Y = Y \setminus X^I$, $X_Y^I = X^I$, *and* $X_Y^O = X^O$.
- *For every* $s_Y \in S_Y$, $Init_Y(s_Y) = true$ *iff* $\exists s \in S.\ (Init(s) = true) \wedge (s_Y = s[X_Y])$.
- *For every* $s_Y, s'_Y \in S_Y$, $T_Y(s_Y, in, s'_Y) = true$ *iff* $\exists s, s' \in S.\ (T(s, in, s') = true) \wedge (s_Y = s[X_Y]) \wedge (s'_Y = s'[X_Y])$.

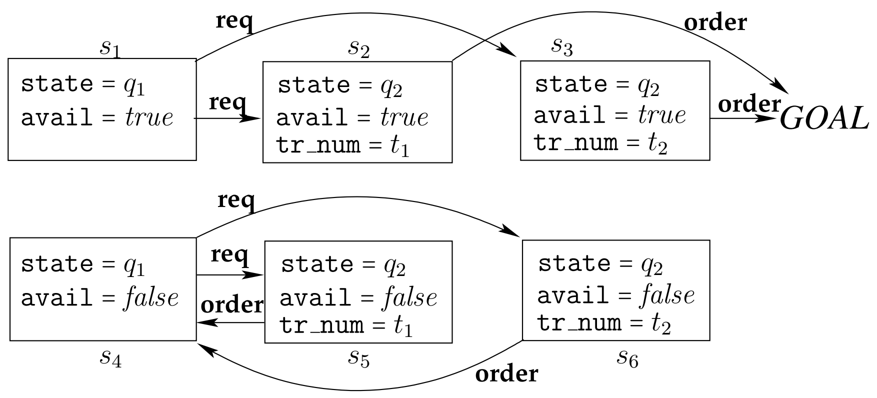
Since W_Y preserves the signature of W, once we construct a coordinator c which can be composed with W_Y based on Definition 10, c also can be composed with W. Moreover, since the abstraction W_Y over-approximates the concrete web services W (i.e., W_Y contains all the behaviors of W), W_Y satisfies the following property.

Theorem 4 (Soundness)**.** *Given a set W of web services and a goal G, if a coordinator web service c satisfies $W' \| c \vDash G$ where W' is a signature-preserving abstraction of W (e.g., W_Y in Definition 15), then c also satisfies $W \| c \vDash G$.*

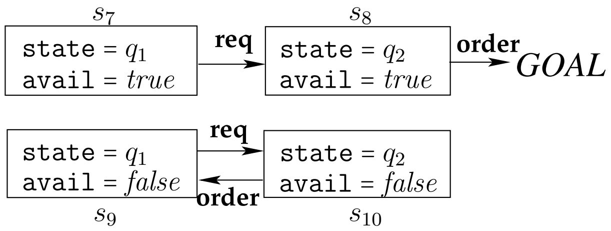
Example 5 (Abstraction)**.** *Figure 4.3(a) illustrates the concrete state space with 6 states, where there are three internal variables—*`state`, `avail`, `tr_num`*. Symbols above arrows represent a value of an input variable. In this example, from the state s_1, we have a strategy to guarantee to reach* GOAL*—invoking* **req** *and* **order**. *Figure 4.3(b) shows an abstract state space with respect to* $\{$`state`, `avail`$\}$. *s_1 and s_4 in the original space are mapped to s_7 and s_9, respectively. Two states, s_2 and s_3, (s_5 and s_6) collapse into s_8 (s_{10}, respectively). Although the number of states decreases, every path in the original state space is mapped to one of paths in the abstract space. Moreover, from the state s_7 corresponding to s_1, we still have a strategy to guarantee to reach* GOAL. *Figure 4.3(c) shows a coarser abstraction. However, from the state s_{11} corresponding to s_1, we no longer have a strategy to guarantee to reach* GOAL *since we abstract out too much.*

4.3.2.2 Abstraction and Refinement algorithm

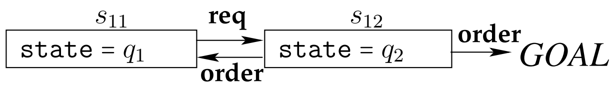
Algorithm 4 presents a high-level description of our method based on signature-preserving abstraction. In a nutshell, we abstract a given web services W into W'



(a) Original states for {state, avail, tr_num}



(b) Abstract states for {state, avail}



(c) Abstract states for {state}

Figure 4.3. Abstraction

and try to find a solution for the abstraction W'. If we identify such a coordinator, it can indeed control the original web services W to satisfy a given goal. Otherwise, we repeat the search with more accurate abstraction.

First, we abstract W with only input and output variables, i.e., $Y = X^I \cup X^O$ (lines 1–2). Since, at this point, W_Y does not include any internal variable (i.e., $X_Y = X_Y^O$), we can exploit, in this case, the algorithm for WSC with full observation, *WSCFullObs*, which is more efficient algorithm (EXP-hard). For the sake of space, we do not show the details of *WSCFullObs*. If we find a coordinator c such that $W_Y \| c \vDash G$, then c also satisfies $W \| c \vDash G$ by Theorem 4. Otherwise, we refine our current abstraction W_Y by adding more variables, and try to find c for the new abstraction (lines 6–10). How to select additional variables will be elaborated in Section 4.3.2.3. We repeat the abstration/refinement step until we identify a coordinator c satisfying $W_Y \| c \vDash G$ or the variable set used for abstraction equals

Algorithm 4: Signature-preserving Abs/Ref WSC

```
Input  : A set $W$ of web services and a goal $G$.
Output: A coordinate web service $c$.
1  $Y := X^I \cup X^O$;
2  $W_Y := Abstraction(W, Y)$;                    // $W_Y$ has only $X^I$ and $X^O$.
3  if $((c := WSCFullObs(W_Y, G)) \neq null)$ then
4      return $c$;
5  end if
6  $ConstructDependencyGraph(W, G)$;
7  while $((newVars := SelectNewVars(W, G)) \neq null)$ do
8      $Y := Y \cup newVars$;
9      $W_Y := Abstraction(W, Y)$;
10     if $((c := WSCPartialObs(W_Y, G)) \neq null)$ then
11         return $c$;
12     end if
13 end while
14 return $null$;
```

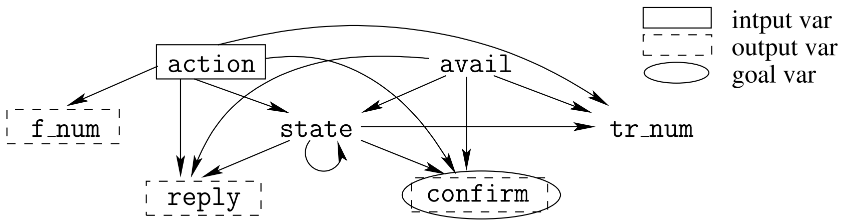


Figure 4.4. Variable dependency graph

to the original variable set. The latter case implies that no solution exists for the given problem. Although from the second loop, we should employ the algorithm for WSC with partial observation, *WSCPartialObs*, with $O(2^{2^n})$ complexity, once we identify a coordinator using small abstract web services, searching space is shrunken (double-)exponentially in the number of variables that we save.

4.3.2.3 Automatic refinement

If we fail to identify a coordinator for abstract web services (line 3 or 9 in Algorithm 4), it is caused either by too coarse abstraction or by the fact that a coordinator for the original web services does not exist. For the latter case, since we check it with the original web services in the worst case, Algorithm 4 will correctly conclude that there is no solution.

Theorem 5 (Completeness)**.** *Given a set of web services W and a goal G, if there does not exist a coordinator c to satisfy $W \| c \models G$, Algorithm 4 eventually returns null.*

However, in the former case, although there exists a coordinator for the original web services W, *WSCFullObs* or *WSCPartialObs* returns *null* for the abstraction W_Y. The reason is that removing too many variables, including ones with significant information to reach a goal, gives too much freedom to the abstraction. It induces some infeasible paths to states not satisfying the goal. For instance, in Figure 4.3(c), since we remove the variable `avail`, s_1 (s_2 and s_3) is indistinguishable from s_4 (s_5 and s_6, respectively). Thus, an infeasible edge from s_{12} to s_{11} by **order** is introduced, by which we no longer have a strategy to guarantee to reach a goal. Therefore, we have to refine the current abstraction to find a solution by adding more variables. Since the infeasible paths to states not satisfying the goal prevent us from identifying a solution coordinator, it is important to accurately keep track of the values of variables appearing in the given goal predicate. With this reason, the most significant criterion for selecting variables to be added is the relevance to variables in the goal predicate. To evaluate each variable's relevance to the goal variables, we construct a variable dependency graph.

Definition 16 (Variable dependency graph)**.** *Given a set of web services W and a goal G, a variable dependency graph is a* directed *graph $G(V, E)$ where a set V of vertexes is $\{x \mid x \in X \cup X^I\}$ and a set E of* directed *edges is $\{(x \triangleright y) \mid x, y \in V$, the value of y depends on the value of $x\}$.*

For instance, the pseudo-codes "`y := x`" and "**if** (`x` = *true*) **then** `y :=0`" imply that the value of y depends on x. Figure 4.4 illustrates a fraction of the variable dependency graph for W and G in Example 4. It shows only variables of w_{AR}. For example, since the values of `state`, `reply` and `tr_num` depend on the values of `state`, `action` and `avail` (see the first part of T in Example 3), we have corresponding directed edges (`state` $\triangleright$ `state`), (`action` $\triangleright$ `state`), (`avail` $\triangleright$ `state`), $\cdots$, and (`action` $\triangleright$ `tr_num`) in Figure 4.4. In the dependency graph, it is clear that variables with stronger dependency to the variables in the goal predicate locate closer to the goal variables. Thus, in each iteration of Algorithm 4, the procedure *SelectNewVars* returns a set of variables that have the closest hop to the variables in the goal predicate (i.e., 1-hop, 2-hop, and so on). For instance, since `confirm` is a variable in the goal predicate, the set of variables that have 1-hop dependency is {`action`, `avail`, `state`}.

4.3.3 Signature-subsuming Abstraction

In Section 4.3.2, we restricted the target of abstraction to internal variables; namely, abstract web services have the same I/O variables with original ones. However, in many cases, we have observed that some of output variables do not provide any important information for a coordinator to decide its move. For instance, the airline reservation web service in Example 3 simply copies the request value (i.e., *req₁* and *req₂*) to the flight number (i.e., *f₁* and *f₂*), and returns it to clients for reference. In this case, even without this output, the coordinator can successfully control given web services to satisfy the goal. Hence, in this section, we consider, as the target of abstraction, output variables as well as internal variables.

First, we define *signature-subsuming abstract web services* for given web services, which have the same input variables, but less internal variables and output variables.

Definition 17 (Signature-subsuming abstract web services)**.** *Given a set of web services $W(X, X^I, X^O, Init, T)$, and a set Y of variables such that $X^I \subseteq Y \subseteq X^A$, the signature-subsuming abstraction of W with respect to Y is $W_Y(X_Y, X^I_Y, X^O_Y, Init_Y, T_Y)$ where $X_Y = Y \setminus X^I$, $X^I_Y = X^I$, $X^O_Y = Y \cap X^O$, and $Init_Y$ and T_Y are defined as the same as Definition 15.*

Since signature-subsuming abstract web services W_Y have less output variables than the original web services W, any coordinator c which can be composed with W_Y is also able to be composed with W by ignoring redundant output variables of W (i.e., ignoring $X^O \setminus X^O_Y$). Moreover, since W_Y contains all the behaviors of W, Theorem 4 is still valid.

For selecting output variables to be used in abstraction, we again employ the variable dependency graph in Section 4.3.2.3. In general, output variables that depend on internal variables that in turn depend on variables in a goal predicate, tend to provide important information on the state of web services for the coordinator to control the web services. For instance, in Figure 4.4, `reply` has

Table 4.2. Experiment result

Problem	Total var	I/O var	Basic	Signature preserving	Saved var	Signature subsuming	Saved var
TAS-a	38	9	5.8	2.9	6	**0.1**	6/4
TAS-b	42	8	61.4	55.3	2	**13.8**	2/1
TAS-c	69	10	>7200.0	>7200.0	6	**162.0**	6/2
P&S-a	44	9	50.4	49.8	11	**3.2**	11/2
P&S-b	55	10	320.0	364.6	19	**42.3**	19/3
P&S-c	63	10	>7200.0	>7200.0	20	**1214.0**	20/3
VOS-a	61	15	208.3	195.7	14	**18.2**	14/4
VOS-b	74	15	3323.0	2321.3	23	**520.8**	23/4

a dependency on `state` and `avail` that have a dependency on the goal variable `confirm`, and `reply` is an important output by which a coordinator infer whether a flight seat is available. On the other hand, `f_num` that represents a flight number has dependency only on an input variable, `action`, and it does not provide any information to help a coordinator. Therefore, we find such a set $X^{SO} \subseteq X^O$ of *significant output variables* which have a dependency on internal variables with a dependency on variables in a goal predicate, and then use X^{SO} for the initial abstraction. That is, in signature-subsuming abstraction, we start $Y := X^I \cup X^{SO}$ as line 1 in Algorithm 4. The rest of output variables (i.e., $X^O \setminus X^{SO}$) are used in the last iteration.

4.3.4 Empirical Validation

We have implemented automatic tools for signature-preserving/signature-subsuming abstraction and refinement. Given a set of web service descriptions in WS-BPEL files, and a goal predicate, our tools automatically construct a coordinator web service which can control the given web services to achieve the goal. To demonstrate that our tools efficiently synthesize coordinators, we compared the basic algorithm [115] and our methods with 3 sets of realistic examples (8 instances); Travel agency system (TAS), Producer and shipper (P&S), and Virtual online shop (VOS). Since there are no public benchmark test sets, we have selected web service examples popularly used in web service composition researches. TAS was explained in Example 4.1.1. We have three instances, TAS-a, TAS-b, and TAS-c, where we have 4, 9, and 16 options, respectively, for input values for flight reservation and hotel reservation each. Producer and shipper (P&S) [115, 76] includes two web services, Producer and Shipper. Producer produces furniture items, and Shipper delivers an item from an origin to a destination. We have three instances, P&S-a, P&S-b, and P&S-c where there are 4, 6, and 8 options, respectively, for furniture order and delivery order each. Virtual online shop (VOS) [117] includes Store and Bank web services where Store sells items and Bank transfers money from one account to another account. This example includes two instances, VOS-a and VOS-b where there are 3 and 4 options, respectively, for item orders and money transfer each.

All experiments have been performed on a PC using a 2.4GHz Pentium processor, 2GB memory and a Linux operating system. Table 4.2 presents the number of total variables (Total var) and input/output variables (I/O var) in boolean. It also shows the total execution time in seconds for the basic algorithm (Basic) [115] and our methods (Signature-preserving and Signature-subsuming), and the number of boolean variables that we saved (Saved var). In the signature-subsuming case, the table presents the number of internal variables/IO variables which we saved. Our experiment shows that our technique outperforms the basic algorithm in terms of execution time.

Although we have employed modest size of examples, our abstraction technique can be useful even for larger size examples since in general, the number of variables which have relevance with goal variables is limited. For instance, with 100 web services with 10 variables each (total 1,000 variables), a goal that users want is often associated with only a fraction of available web services and variables (say 5%, 50 variables). In such a case, our techniques can eliminate 95% of irrelevant variables, improving the convergence speed considerably.

Chapter 5

QoS-aware Web Service Composition

Recently, the *web service composition* (*WSC*) requires discovering service providers that satisfy not only functional requirements but also nonfunctional ones, including *Quality of Services* (*QoS*) constraints. In this case, one indeed desires *not* the shortest sequence of web services for a solution *but* the composite web service with the optimal accumulated QoS value. We call it *QoS-aware web service composition* (*QoS-aware WSC*) problem. It is computationally hard to identify such a composite web service since this problem is reduced to a global optimization problem. However, many engineering tasks in the real world require real-time responsiveness, and the users do not allow a long delay to algorithms. Furthermore, if the size of a given set of web services is large, like real world web services on Web, the problem is intractable.

To resolve this challenge, we propose to apply *anytime algorithm* [16] to the QoS-aware WSC problem. While traditional algorithms provide only one answer after completely terminating a fixed number of computations, anytime algorithms can return many possible approximate answers to any given input. They always provide available best-so-far answers and the quality of the answer improves along with execution time. If enough time is allowed, the algorithm will provide the optimal answer eventually. By using an anytime algorithm for the QoS-aware WSC problem, we can identify a composite web service with good quality earlier than optimal algorithms. To the best of our knowledge, there is no work to employ anytime algorithm to the WSC problems.

Our anytime algorithm proposed in this dissertation is based on *beam stack search* [17] which explores, in each level, a fixed number of candidate states (called *beam width*). In this searching mechanism, a bad decision in early phases requires more searching space than a bad selection in late phases do. To reduce this cost, our proposal *dynamically* assigns a larger beam width to early phases so that we can consider more qualified candidates. In addition, we propose two more heuristics to improve the quality of current approximate solutions and overall execution time. We validate our proposal with experiment on a number of examples that are generated by the web service testset generator for Web Services Challenge 2009 [18].

5.1 Preliminaries

Quality of Services (QoS) is considered as several qualities or properties of a service, such as *response time* (the time a service takes to respond to various types of requests) and *throughput* (the rate at which a service can process requests). As a natural price of a web service, QoS helps clients (applications as well as human being) to select a service provider with good quality. Moreover, enhanced QoS of web services will bring a highly competitive advantage for web service providers. This implies that users and providers need to be able to engage in QoS negotiation. As a result, recently new standards for the specification of QoS requirements and service-level agreements of a single web service, such as *WS-Policy* [13], the *Web Service Level Agreement (WSLA) language* [14] and *WS-Agreement* [15], have been introduced.

Now, given a set of web services with their QoS values, QoS-aware WSC problem is to find out a sequence of web services with the optimal aggregated QoS value, which is a path from the initial state to a given goal state. We reduce the problem into a well known planning problem on a weighted graph such that the optimal solution of the planning problem always corresponds to the optimal solution of our QoS-aware WSC problem.

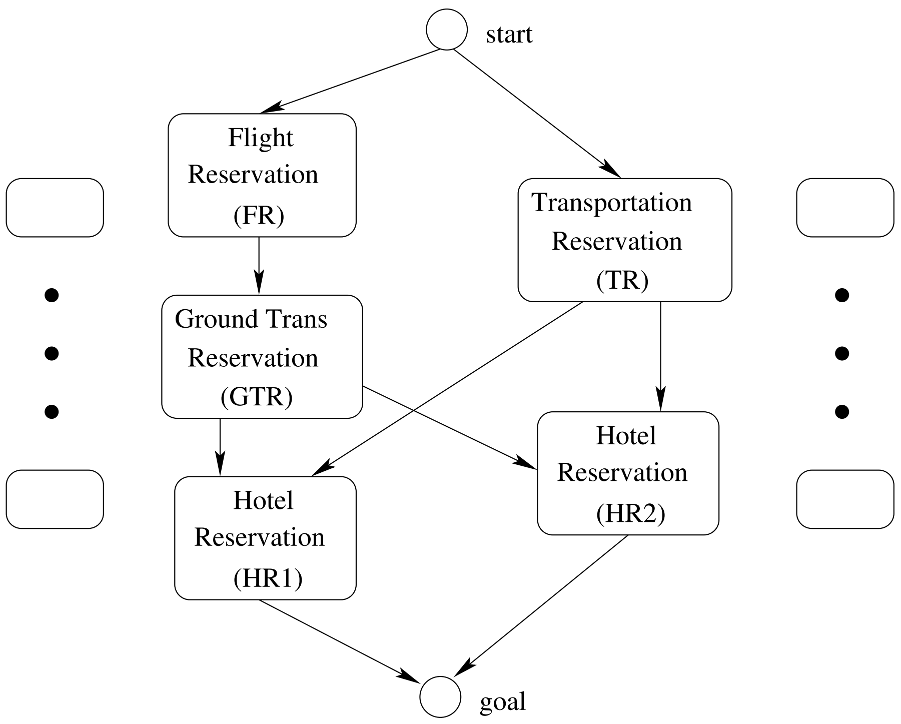


Figure 5.1. Travel agency system

5.1.1 Example: QoS-based Travel agency system

Consider a client wants to make a reservation for a trip including flights, ground transportation and a hotel with the *fastest response time*. However, since there are many reservation web services with similar functionality but different response time, the client wants to combine required web services with considering aggregated response time of a composite service. Assume that a user provides information necessary for the trip, including departing/returning dates, an origin and a destination.

Fig 5.1 illustrates this example. Transportation Reservation (TR) service deals with the reservation for all public transportation including flights and ground transportation. On the other hand, Flight Reservation (FR) service and Ground Trans Reservation (GTR) service treat a request only for flights and ground transportation, respectively. Hotel Reservation (HR1 and HR2) services process the reservation for hotel rooms. The response time for each web service is as follows: $R_{TR} = 100\,msec$, $R_{FR} = 20\,msec$, $R_{GTR} = 70\,msec$, $R_{HR1} = 90\,msec$, and $R_{HR2} = 100\,msec$, respectively. In this example, we have four sequences to make reservations for all of flights, ground transportation, and a hotel room: i.e., FR-GTR-HR1, FR-GTR-HR2, TR-HR1, and TR-HR2. If considering the length of composite web services as our aim, TR-HR1 and TR-HR2 would be the best compositions. However, in this example based on response time, FR-GTR-HR1 is the optimal composition since it has the minimal response time as $R_{FR-GTR-HR1} = 180\,msec$.

5.1.2 QoS-aware Web Service Composition Problem

In this section, we formalize the notion of web services with QoS criteria and their QoS-aware composition that we consider in this dissertation. A *web service* is a tuple $w(I, O, Q)$ with the following components:

- I is a finite set of *input parameters* for w.
- O is a finite set of *output parameters* for w; each input/output parameter $p \in I \cup O$ has a type t_p.
- Q is a finite set of *quality criteria* for w.

When a web service $w(I, O, Q)$ is invoked with all the input parameters $i \in I$ with the type t_i, it returns all the output parameters $o \in O$ with the type t_o. The invocation of the web service w corresponds to each service quality criterion $q \in Q$, for instance, a response time q_r or a throughput q_t. We assume that the signature description for each web service is given in *Web Services Description Language (WSDL)* and the quality criteria of a web service is given in *Web Service Level Agreements (WSLA)*. Given two types t_1 and t_2, t_1 is a *subtype* of t_2 (denoted by $t_1 <: t_2$) if t_1 is more informative than t_2 so that t_1 can substitute for t_2 everywhere. In this case, t_2 is a *supertype* of t_1. This relation is reflexive (i.e., $t <: t$ for any type t) and transitive (i.e., if $t_1 <: t_2$ and $t_2 <: t_3$ then $t_1 <: t_3$). We assume that the type hierarchy is given; e.g., specified in OWL. Given two web services $w_1(I_1, O_1, Q_1)$ and $w_2(I_2, O_2, Q_2)$, we denote $w_1 \sqsupseteq_I w_2$ if w_1 requires more informative inputs than w_2; i.e., for every $i_2 \in I_2$, there exists $i_1 \in I_1$ such that $t_{i_1} <: t_{i_2}$. Given two web services $w_1(I_1, O_1, Q_1)$ and $w_2(I_2, O_2, Q_2)$, we denote $w_1 \sqsubseteq_O w_2$ if w_1 provides less informative outputs than w_2; i.e., for every $o_1 \in O_1$, there exists $o_2 \in O_2$ such that

$t_{o_2} <: t_{o_1}$. A *web service discovery problem* is, given a set W of available web services and a request web service w_r, to find a web service $w \in W$ such that $w_r \sqsupseteq_I w$ and $w_r \sqsubseteq_O w$.

However, it might happen that there is no single web service satisfying the requirement. In that case, we want to find a sequence $w_1 \cdots w_n$ of web services such that we can invoke the next web service in each step and achieve the desired requirement eventually. Formally, we extend the relations, $\sqsupseteq_I$ and $\sqsubseteq_O$, to a sequence of web services as follows.

- $w \sqsupseteq_I w_1 \cdots w_n$ (where $w = (I, O, Q)$ and each $w_j = (I_j, O_j, Q_j)$) if $\forall 1 \leq j \leq n$: for every $i_2 \in I_j$ there exists $i_1 \in I \cup \bigcup_{k<j} O_k$ such that $t_{i_1} <: t_{i_2}$.

- $w \sqsubseteq_O w_1 \cdots w_n$ (where $w = (I, O, Q)$ and each $w_j = (I_j, O_j, Q_j)$) if for every $o_1 \in O$ there exists $o_2 \in \bigcup_{1 \leq j \leq n} O_j$ such that $t_{o_2} <: t_{o_1}$.

Finally, given a set W of available web services and a service request w_r, the *QoS-aware web service composition problem* $\langle W, w_r \rangle$ which we focus on in this thesis is to find a sequence $w_1 \cdots w_n$ (every $w_j \in W$) of web services such that $w_r \sqsupseteq_I w_1 \cdots w_n$ and $w_r \sqsubseteq_O w_1 \cdots w_n$. The optimal solution for this problem is such a sequence $\sigma = w_1 \cdots w_n$ to minimize the aggregate QoS value $Q(\sigma)$. Given a sequence $\sigma = w_1 \cdots w_n$, the aggregate QoS value $Q(\sigma)$ is computed as follows:

- $Q(\sigma) = c_1 \cdot Q_1(\sigma) + \cdots + c_m \cdot Q_m(\sigma)$ where each c_i is a given weight for the i-th quality criterion.

- In the above, each function Q_i depends on the corresponding quality criterion. For instance, assume that the quality criterion is response time. If $|\sigma| = 1$ (i.e., $\sigma = w$), then $Q_i(\sigma) = rt_w$ where rt_w is the response time of w. Otherwise (i.e., $|\sigma| > 1$), $Q_i(\sigma) = rt_{w_1} + Q_i(w_2 \cdots w_n)$. On the other hand, consider throughput as the quality criterion. If $|\sigma| = 1$ (i.e., $\sigma = w$), then $Q_i(\sigma) = th_w$ where th_w is the throughput of w. Otherwise (i.e., $|\sigma| > 1$), $Q_i(\sigma) = \mathrm{Min}(th_{w_1}, Q_i(w_2 \cdots w_n))$.

5.1.3 Reduction to a planning problem

Given a QoS-aware web service composition problem $\langle W, w_r \rangle$, the problem can be reduced into a planning problem on a weighted state-transition system. A *weighted* *state-transition system* is a 6-tuple $\mathcal{S} = (S, s_0, F, A, T, C)$ with the following components:

- S is a finite set of *states*.
- $s_0 \in S$ is an *initial states*.
- $F \subseteq S$ is a set of *final states*.
- A is a set of *actions*.
- $T : S \times A \rightarrow S$ is a *transition function*.
- For each state s and each action a, $C(s, a)$ is the action cost.

The planning problem is to find a sequence of actions from the initial state to a final state which minimizes the total cost. The solution can be represented in terms of Bellman equation which characterizes the *optimal cost function* [118]:

$$V(s) = \begin{cases} 0 & \text{if } s \text{ is a final state} \\ \min_{a \in A} Q_V(s, a) & \text{otherwise} \end{cases}$$

where $Q_V(s, a) = C(s, a) + V(s')$ and $s' = T(s, a)$.

There exists a unique optimal value function $V^*(s)$ that solves the Bellman equation, and the optimal solution can be represented as a strategy σ. The strategy σ is a function mapping a state $s \in S$ into an action $a \in A$.

Given a set $W = \{w_1, \cdots, w_n\}$ of web services where for each j, $w_j = (I_j, O_j, Q_j)$, we denote as TP a set of all types t such that there exists $p \in \bigcup(I_j \cup O_j)$ and t is the type of p. Then, given a set $W = \{w_1, \cdots, w_n\}$ of web services and a requirement web service $w_r(I_{w_r}, O_{w_r}, Q_{w_r})$, we can construct a weighted state-transition system $\mathcal{S} = (S, s_0, F, A, T, C)$ as follows:

- $S = \{(x_1, \cdots, x_m) \mid x_j = \textit{true} \text{ or } \textit{false}, m = |TP|\}$; each boolean variable x_j represents whether we have an instance with the type $t_j \in T$ at a state s.
- $s_0 = (x_1, \cdots, x_m)$ where each x_j is *true* if there exists an input parameter $i \in I_{w_r}$ such that its type t_{x_j} is a supertype of t_i (i.e., $t_i <: t_{x_j}$); otherwise x_j is *false*.

- $F = \{(x_1, \cdots, x_m) \mid$ each x_j is *true* iff there exists an output parameter $o \in O_{w_r}$ such that its type t_{x_j} is a subtype of t_o (i.e., $t_{x_j} <: t_o$) $\}$.

- $A = W$.

- For $s = (x_1, \cdots, x_m)$, $s' = (x'_1, \cdots, x'_m)$ (each x_k and x'_k are *true* or *false*), and $w_j = (I_j, O_j, Q_j)$, $T(s, w_j) = s'$ iff (1) for every $i \in I_j$, there exists x_k in s such that x_k is *true* and its corresponding type t_{x_k} is a subtype of the type of i (i.e., $t_{x_k} <: t_i$), (2) if x_l is *true*, x'_l is also *true*, and (3) $\forall o \in O_j$: for every variable x'_k in s', if its corresponding type t_{x_k} is a supertype of t_o, x'_k is *true*. Intuitively, if a web service w_j is invoked at a state s where we have data instances being more informative than inputs of w_j, we proceed to a state s' where we retain all the data instances from s and acquire outputs of w_j as well as their supertypes.

- For every state s, $C(s, a) = c_1 \cdot q_1 + \cdots + c_m \cdot q_m$ where each q_j is given from w which is corresponding to a. We assume that each weight constant c_j is given.

Intuitively, we have an initial state where we possess all the data instances corresponding to the input of w_r as well as ones corresponding to their supertypes. As goal states, if a state is more informative than the outputs of w_r, it is a goal state. Finally, given a QoS-aware web service composition problem $\langle W, w_r \rangle$, we can reduce it into a planning problem on $\mathcal{S} = (S, s_0, F, A, T, C)$ where the optimal cost path from an initial state to a goal state corresponds to the optimal composition of web services.

5.2 Anytime Algorithm for QoS-aware WSC

Most algorithms provide only one solution after completely terminating a fixed number of computations. Sometimes, its required computation number can be so large that it takes a long time until completion. Even, it is often for algorithms to pass a given time-out with blaming a lack of system resource after a long waiting. However, many engineering tasks in the real world require real-time responsiveness. Furthermore, users do not allow a long delay to algorithms. Considering such a practical issue, we prefer a solution with acceptable quality (e.g., the answer falls within the range of tolerance) to the optimal solution requiring a long or an infinite time if an algorithm can identify it within a given time-out. Anytime algorithm were designed to conform to this requirement.

An anytime algorithm always provides a best-so-far answer available although the answer may be an approximation of the optimal answer. The quality of the answer improves with the execution time, and if enough time is allowed, the answer will be converged to the optimal one eventually. While, in most of cases, it takes very short time for anytime algorithms to generate the first approximate answer, anytime algorithms require longer time to find the optimal answer than optimal algorithms do since they explore more searching space to generate approximate solutions earlier. The user may examine the answer at anytime, and choose to terminate with satisfaction or to continue the execution for a better answer. Due to these capabilities, anytime algorithm has been successfully used in AI and real-time system literatures.

In this dissertation, we propose an anytime algorithm for QoS-aware WSC problem, which is a global optimization problem and requires fast response time. Our anytime algorithm is based on the beam stack search [17] that has shown an excellent performance in AI planning literature. However, since the beam stack search assigns a fixed value to the beam width for each search level, it considers the same number of candidates in all the steps, which is not nature. Based on this observation, we dynamically adjust the beam width to consider more candidates in earlier steps than in later steps. In addition, we propose more heuristics to improve the performance of our algorithm.

5.2.1 Basic Algorithm: Beam Stack Search

The *beam stack search algorithm* [17] is an anytime search algorithm based on the *beam search*. We first introduce the beam search in this section.

The *beam search* is a well-known approximate search algorithm which explores *not* all the nodes *but* only a set of candidate nodes in each search level. Algorithm 5 illustrates how the beam search builds its search graph. While any new node to be searched (in *open*[*level*]) exists, it repeats to pick up a node with the minimum cost

Algorithm 5: Beam Search

```
Input  : A initial node $start$ and a goal node $goal$.
Output: A sequences of nodes $sn$.
$open[0] := start$;
$closed[0] := null$;
$level := 0$;
while $(open[level] \neq null) \vee (open[level+1] \neq null)$ do
    while $open[level] \neq null$ do
        $node := argmin\{f(n) \mid n \in open[level]\}$;
        move $node$ from $open[level]$ to $closed[level]$;
        if $(node\ is\ goal)$ then
            return $solutionReconstruction(node)$;
        end if
        $node.generateSuccessors()$;
        if $beamWidth < layerSize(level+1)$ then
            $pruneLayer(level+1)$;
        end if
    end while
    $level := level + 1$;
    $open[level+1] := null$;
    $closed[level] := null$;
end while
return $null$;
```

Algorithm 6: pruneLayer() in Beam Search

```
Input: A level of a search graph $l$
$keep :=$ the best $w$ nodes $\in open[l]$;
$prune := \{n \mid n \in open[l] \wedge n \notin keep\}$;
for each $n \in prune$ do
    $open[l] := open[l] \setminus \{n\}$;
    delete $n$;
end for
```

among the left open nodes at the current level (into *closed*[*level*]), and to check whether it is a goal. If the node is a goal, the algorithm returns the path from a start node to this node as a solution; otherwise, it expands the node (Algorithm 5). To reduce its search space, the algorithm leaves only the fixed number of nodes at each level of the search graph as promising candidates, and permanently prunes off the other nodes (Algorithm 6). We call the set of most promising search nodes per a level as *beam* and the size of the set as *beam width*, respectively. Note that beam searches assign the same number as the beam width for every level. Since using a fixed beam width makes the space and time complexity of the beam search become linear in the depth of the search instead of exponential, the beam search is widely applied to problems requiring huge search spaces.

However, its space-limited search induces a serious drawback—incompleteness. By pruning the nodes inadmissibly without any rationale, it may miss the optimal solution. Indeed, a beam search is a restricted version of a breadth-first search in the sense that the size of the set of search nodes at each level is limited and non-promising nodes can be pruned at any step in the search. With an unlimited beam width, beam search behaves like breadth-first search which can search completely. However, in this case, its space requirement is exponential with the depth of the search.

To guarantee a complete search upholding efficiency of beam search, the *beam stack search algorithm* prunes *admissibly* the search space based on *upper* and *lower bounds*. Algorithm 7 describes how the beam stack search works. The beam stack search has three inputs, i.e., *start*, *goal* and U. Each *start* and *goal* indicates the starting and the terminal node of a search graph where a node represents a state in a planning problem. A given U is an upper bound of an acceptable aggregated QoS value. Therefore, a solution generated by the beam stack search does not have an aggregated QoS value greater than U. Besides, each time that beam-stack search finds an improved solution of which aggregated QoS value is lower than the current upper bound U, it updates the current upper bound with the aggregated QoS value of the current solution.

The basic framework of the beam stack search is similar with beam search; i.e., the algorithm repeats to pick up the most promising node from the current open nodes and to check if the node is goal, until no more *qualified* node at the current level or the next level exists. If it is a goal, the algorithm returns the path from a start node to this node. Otherwise, the algorithm produces its *qualified* successors, keeping a beam width at the next level. The main difference from the beam search is a condition for a *qualified* node. In the beam search (let us assume that the beam width is k), a qualified node in a beam should have an aggregated QoS value which is in the top k ranking. In the beam stack search, in addition to the above requirement, the aggregated QoS value of a qualified node should be in the range

Algorithm 7: Beam Stack Search

```
Input  : A initial node $start$, a goal node $goal$ and an allowedUpperCost $U$.
Output: A sequences of web services $ws$.
open[0] := start;
closed[0] := null;
bestGoal := null;
level :=0;
while $(open[level] \neq null) \lor (open[level+1] \neq null)$ do
    while $open[level] \neq null$ do
        $node := argmin\{f(n) \mid n \in Open[level]\}$;
        Move $node$ from $open[level]$ to $closed[level]$;
        if $(node\ is\ goal)$ then
            $bestGoal := node$;
            $U := g(node)$;
        end if
        $node.generateAdmittedSuccessors(beamstack.top())$;
        if $beamWidth < layerSize(level+1)$ then
            $pruneLayer(level+1)$;
        end if
    end while
    if $(1 < level \leq relay) \lor (relay + 1 < level)$ then
        delete $closed[level-1]$;
    end if
    $level := level + 1$;
    $open[level+1] := null$;
    $closed[level] := null$;
    $beamstack.push([0, U])$;
end while
if $bestGoal() \neq null$ then
    return $solutionReconstruction(best - goal)$;
end if
else return $null$;
```

of the current lower and upper bounds. These bounds are stored into the *beam stack* and they represent what interval we have explored in each level of a search graph. Each entry of a beam stack is $[f_{min_i}, f_{max_i})$ where i is the index of the beam stack and it corresponds to the i-th level of search graph. The lower and upper bounds represent that at the corresponding level, the algorithm has searched all nodes with the QoS value which is equal or greater than f_{min} and is less than f_{max}. At the beginning, all f_{min} and f_{max} are initially assigned as 0 and U, respectively.

Algorithm 8: PruneLayer() in Beam Stack Search

```
Input: A level of a search graph $l$
1  $keep$ := the best $w$ nodes $\in open[l]$;
2  $prune := \{n \mid n \in open[l] \wedge n \notin keep\}$;
3  $beamstack.top().f_{max} := argmin\{f(n) \mid n \in prune\}$;
4  for each $n \in prune$ do
5      $open[l] := open[l] \setminus \{n\}$;
6      delete $n$;
7  end for
```

Algorithm 9: QoS-aware WSC with Beam Stack Search

```
Input  : A set $W$ of web services, a web service $w$ and an allowedUpperCost $U$.
Output: A set of sequences of web services $sws$.
1  Make $start$ and $goal$ from $w$;
2  $initializeBeamStack(U)$;
3  $solPath_{opt} := null$;
4  while $beamstack.top() \neq null$ do
5      $solPath := BSSearch(start, goal, U)$;
6      if $solPath \neq null$ then
7          $solPath_{opt} := solPath$;
8      end if
9      while $beamstack.top().f_{max} \geq U$ do
10         $beamstack.pop()$;
11     end while
12     if $beamstack.isEmpty()$ then
13         return $solPath_{opt}$;
14     end if
15     $beamstack.top().f_{min} := beamstack.top().f_{max}$;
16     $beamstack.top().f_{max} := U$;
17 end while
```

If no node was inadmissibly discarded at the level, these values do not change. However, if a node is removed due to a limited size of beam width, the beam stack keeps the minimum QoS value of pruned nodes at the level as f_{max} to remember the best discarded node at the current search iteration (Algorithm 8). At the next search iteration, as candidates at that level, we may pick up the nodes which were pruned at the previous iteration but are necessary to be searched. These nodes have QoS values at least equal to or greater than f_{max}.

Next, Algorithm 9 shows how to solve the QoS-aware WSC problem by using the beam stack search. It first generates a *start* and a *goal* state from a given target web service w, and initializes the necessary variables (lines 1–3). While any entry of a beam stack exists, the algorithm tries to find a better answer of which aggregated QoS value is lower than the current U by calling the beam stack search function, `BSSearch()`, in Algorithm 7 (lines 4–17). Note that since this algorithm provides the *start* state to a beam stack search function as an input, the search always starts from the root node at the search graph (line 5). When a beam stack search function terminates, the beam stack is checked to trace if any node was inadmissibly discarded from the deepest level at the previous search stage, which corresponds to the top of the beam stack. If no node was inadmissibly discarded (i.e., f_{max} of beam stack is equal to or greater than U), then the algorithm repeats popping out the top item of the beam stack until a top item has a f_{max} less than U (lines 9–11). After this process, if the beam stack is empty, it means that the algorithm searches completely and the best-so-far answer is an optimal solution (lines 12-14). Otherwise, it updates the current lower and upper bounds at the top item of a beam stack as $[f_{max}, U)$ (lines 15–16). With this updated record, at the next search stage, the algorithm can go back to that level and hold on the nodes of which costs are at least as bad as the best previously discarded node but may be less than the cost of the current solution.

5.2.2 Beam Stack Search with Dynamic Beam Width

A beam stack search explores, in each level, a *fixed* number of nodes based on the range of QoS values at a beam stack item. In this searching mechanism, a bad decision in early phases requires more searching space than a bad selection in late phases does. To reduce this cost, in section 5.2.2.1, we propose to *dynamically* assign larger beam widths to early phases to consider more qualified candidates for a better solution. Moreover, we propose two more heuristics in its implementation to improve the quality of current approximate solutions and overall execution time in section 5.2.2.2. Algorithm 10 and Algorithm 11 illustrate how to solve QoS-aware WSC problem with our proposed heuristics based on the basic algorithm.

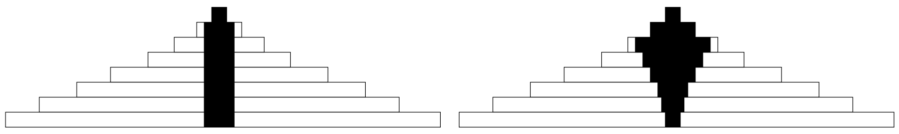


Figure 5.2. Search with static beam width vs. dynamic beam width

5.2.2.1 Carrot-shaped Dynamic Beam Width

An original beam stack search employs a static beam width at all levels of a search graph regardless of a given problem. However, the size of the beam width plays a significant role in determining the efficiency of beam stack search. A small sized beam width causes excessive pruning of qualified successor nodes, and therefore it requires more extensive backtracking and node re-expansion. On the other hand, with a large sized beam width, many candidate nodes will be under consideration, and it makes a searching space increased exponentially. In fact, with an extremely small or large size of beam width, the beam stack search is no better than depth-first search and breadth-first search. Therefore, we need to consider an efficient method to control the size of beam width.

To resolve this issue, we propose to dynamically adjust a beam width; we assign larger beam widths to early stages to consider more candidate nodes. This idea is influenced by a following natural intuition. When finding out an unknown path from a source to a destination, we do not know whether we have to go north, south, east or west, especially at the beginning. A good choice at early stage leads to a good solution path more quickly. A bad decision in early stages requires a longer backtracking and many retrials to other paths, while a wrong decision in late stages induces just a short backtracking. Hence, selections at early stages are more critical than late stages. In the beam stack search, the benefit and cost at earlier levels are also significant due to its backtracking mechanism based on a beam stack. Once candidate nodes are determined at a level, the other nodes excluded cannot be visited until completing whole searches for the subtrees of the selected candidates.

In this dissertation, we propose to apply different sized beam widths during searching, that is, our technique uses a large sized beam width first and decreases its size gradually as going down to a search graph. As a result, a constructed search graph for one search iteration appears like a carrot in shape (in Figure 5.2).

To determine the beam width values, we should consider various features. Among them, the number of all nodes in each level of a search graph is one of main factors to determine the size of the searching space. To exactly identify the number of nodes in each level, we should construct the whole search graph, and this work requires a significant amount of computations. Therefore, by assuming that every node has a similar number of successors, we can approximate these values. In the level 0, we have only one node which is the initial node. We then compute the number of successors of the initial node called `numSucc`. We approximate the number of successors of each node as this value, `numSucc`. Now, in the level 1, we have exactly `numSucc` nodes, and there are approximately $\texttt{numSucc}^2$ nodes, $\texttt{numSucc}^3$ nodes, $\texttt{numSucc}^4$ nodes, $\cdots$ in the level 2, the level 3, the level 4, $\cdots$, respectively. Based on the above approximation, we decide the beam width $beamWidth(i)$ for each level i as following:

- $beamWidth(1) = \texttt{numSucc}$. We consider all nodes for the most careful choice at the highest level.

- $beamWidth(2) = R_{inc} \cdot beamWidth(1)$, where R_{inc} is the increasing rate for the level 2 ($1 < R_{inc} < \texttt{numSucc}$); in our experiment, we take 1.5 as the decreasing rate. In the 2nd level, since the approximate number of nodes is $\texttt{numSucc} \times \texttt{numSucc}$, for a linear complexity we need to restrict the beam width. However, we do not decrease the beam width but increase it linearly by a user provided constant R_{inc}. This strategy is helpful to find a short-length composition.

- For $i \geq 3$, $beamWidth(i) = R_{dec} \cdot beamWidth(i-1)$, where R_{dec} is the decreasing rate ($0 < R_{dec} < 1$); in our experiment, we take 0.9 as the decreasing rate. Since choices at deeper levels are not relatively significant, we decrease the beam width by using a user provided constant R_{dec}.

- $width_{min}$ is the minimum value of a beam width where $0 < width_{min} < \texttt{numSucc}$; in our experiment, we take $0.3 \cdot \texttt{numSucc}$ as the minimum value. Since we decrease the size of a beam width along with the level, it may be down to 0 without the minimum restriction. Besides, if a beam width is too small, it may generate excessive backtracking and retrial. Finally, for every

i, if $beamWidth(i) < width_{min}$, then $beamWidth(i) = width_{min}$.

5.2.2.2 Additional Heuristics

In addition to dynamic beam width, we propose two more heuristics to improve the basic anytime algorithm: *short backtracking* and *upper bound propagation* to f_{max}. By avoiding the duplicated state expansion and unnecessary state inclusion, these techniques help to efficiently find approximate solutions with better QoS values.

Short backtracking heuristic

One of the main aims of the beam stack search is to save the memory consumption [119]. This technique keeps in memory only four levels of the search graph which are most recently considered. By this fact, the level to which the algorithm has to backtrack may be removed from memory, and in this case, the algorithm must recover the missing level where we want to backtrack. Unfortunately, the construction of the target level requires us to construct all the upper levels of the target level. In the other word, the algorithm must return to the initial node and construct again successor nodes at each level until the target level is rebuilt.

To avoid this problem, we propose a short backtracking technique. Our search function, `DBSSearch_H` (see Algorithm 10) and QoS-aware WSC algorithm `QoSWSC_DH` (see Algorithm 11) include this technique based on basic algorithms (Algorithm 7, 8 and 9). Our search function, `DBSSearch_H`, stores in memory sets of the closed states for all visited levels, unlike the basic search function where all closed sets are removed before searching the next level (line 23 in Algorithm 7). At the next invocation to `DBSSearch_H`, our WSC algorithm passes the level corresponding to `beamstack[top-1]` to `DBSSearch_H` as an input parameter (line 20 and line 6 in Algorithm 11), where the level is the parent level of the deepest level in which inadmissible pruning occurs. Then, our heuristic search function `DBSSearch_H` easily obtains the open set of the target level where we should backtrack by simply moving the nodes in the closed set to the open set (lines 5–7 in Algorithm 10), and resumes searching at the level corresponding to `beamstack[top]` by generating new successors according to the updated minimum and maximum values of `beamstack[top]`.

Algorithm 10: DBSSearch_H: Dynamic Beam Stack Search with Heuristics

```
Input  : A initial node $start$, a goal node $goal$ and an allowedUpperCost $U$.
Output: A sequences of web services $ws$.
1  $open[0] := start$;
2  $closed[0] := null$;
3  $bestGoal := null$;
4  $beamWidth := determineBeamWidth(level)$;
5  if $closed[level] \neq null$ then
6      Move $all\ nodes$ of $closed[level]$ to $open[level]$;
7  end if
8  while $(open[level] \neq null) \vee (open[level+1] \neq null)$ do
9      while $open[level] \neq null$ do
10         $node := argmin\{f(n) \mid n \in Open[level]\}$;
11         Move $node$ from $open[level]$ to $closed[level]$;
12         if $(node\ is\ goal)$ then
13             $U := g(node)$;
14             $bestGoal := node$;
15         end if
16         $node.generateAdmittedSuccessors(beamstack.top())$;
17         if $beamWidth < layerSize(level+1)$ then
18             $pruneLayer(level+1)$;
19         end if
20     end while
21     $level := level + 1$;
22     $beamWidth := determineBeamWidth(level)$;
23     $open[level+1] := null$;
24     $beamstack.push([0, U])$;
25 end while
26 if $bestGoal() \neq null$ then
27     return $solutionReconstruction(best - goal)$;
28 end if
29 else return $null$;
```

The size of memory required for our heuristic search function is `SoN`·`BW`·`DL` where `SoN` is the size of a data structure for a node, `BW` is a beam width, and `DL` is the deepest level we have explored during a search. Since the size of memory required is linear in a beam width and a solution length, we believe that it is manageably large.

Algorithm 11: QoSWSC_DH: QoS-aware WSC using Dynamic Beam Stack Search with Heuristics

```
Input  : A set $W$ of web services, a web service $W_r$, an allowedUpperCost $U$,
         initBW_ratio $\alpha_{BW}$, decreaseBW_ratio $\beta_{BW}$ and minBW_ratio $\gamma_{BW}$.
Output: A set of sequences of web services $sws$.
1  Make $start$ and $goal$ from $W_r$;
2  $initializeBeamStack(U)$;
3  $solPath_{opt} := null$;
4  $level := 0$;
5  while $(beam - stack.top() \neq null)$ do
6      $solPath := DBSSearch\_H(start, goal, U, level)$;
7      if $solPath \neq null$ then
8          $solPath_{opt} := mathitsolPath$;
9          return $solPath_{opt}$;
10     end if
11     while $beamstack.top().fmax \leq U$ do
12         $beamstack.pop()$;
13     end while
14     if $beamstack.isEmpty()$ then
15         return $solPath_{opt}$;
16     end if
17     $beamstack.top().fmin := beamstack.top().fmax$;
18     $beamstack.top().fmax := U$;
19     $UpdateBiggerf_{max}(U)$;
20     $level := beamstack.size() - 1$;
21 end while
```

Upper bound propagation

Each entry of the beam stack contains f_{min} and f_{max} which represent a set of states under consideration at the corresponding level; i.e., a qualified state at the level should have an aggregated QoS value in between f_{min} and f_{max}. These values f_{max} as upper bounds, have a significant influence to the size of the search space; i.e., if the value is closer to the optimal value, the algorithm can ignore more states of which the aggregation QoS value is greater than the upper bound f_{max}.

In the basic algorithm, we have observed an inefficient implementation. Whenever a new better solution path is found, the basic algorithm updates only the aggregated QoS value of the so-far-best solution U, but the value for f_{max} at every level remains as the same. Without updating a tight low value for f_{max}, we may

Table 5.1. Experiment problems for QoS-aware WSC algorithm

Problem	Num. of web services	Num. of parameters	Solution path length
P1	50	1000	7
P2	100	10000	5
P3	100	10000	7
P4	500	10000	4
P5	1000	10000	5
P6	1500	10000	6

redundantly explore unnecessary states. Such states more likely exist at higher levels since the aggregated QoS value of a state in high levels is relatively small. Its effect, however, can be significant in a whole searching space since the states in higher levels include a number of descendants in their subtrees.

Therefore, once we identify a new better solution in a search iteration, our method updates the value of f_{max} for every level with the aggregated QoS value of the so-far-best solution U (line 19 in Algorithm 11).

5.3 Experiment

We have implemented two automatic tools implementing the algorithms in Section 5.2, i.e., a QoS-WSC tool with the beam stack search and a QoS-WSC tool with dynamic beam width. Given a WSDL file for a set of web services, an OWL file for the type information, a WSLA file for the QoS information, and a WSDL file for a requirement web service, our tools reduce the QoS-driven WSC problem into a planning problem, and then compute an optimal strategy. Finally, our tools translate the optimal strategy into a BPEL file for the optimal composition.

To demonstrate that our tools efficiently identify an optimal solution, we have experimented on six WSC problems (i.e., P1, P2, ···, P6) produced by the TestSetGenerator which the Web Service Challenge 2009 competition [18] provides. Table 5.1 shows, for each problem, the number of total web services and total parameters, and the length of the optimal solution. All experiments have been performed on a PC using a 2.33GHz Core2 Duo processor, 1GB memory and a Linux operating system.

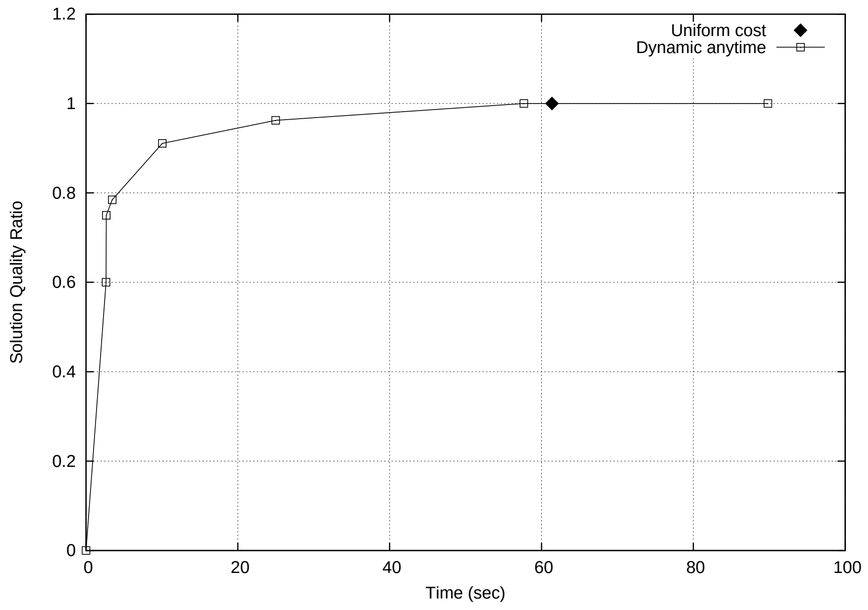


Figure 5.3. Optimal algorithm vs. anytime algorithm on QoS-aware WSC

First, we shows how our anytime algorithm works in solving the QoS-WSC problem, P4, comparing an optimal algorithm. As an optimal algorithm, we employ the uniform cost search algorithm [120], and we compare it with our dynamic anytime algorithm with our proposed heuristics described in section 5.2.2. Figure 5.3 illustrates this comparison, where X-axis is a time in seconds when an algorithm returns a solution and Y-axis is the solution quality ratio of each returned solution (e.g., the QoS value of the optimal solution over the QoS value of a solution returned by an anytime algorithm). Since the uniform cost algorithm is optimal algorithm, it returns a solution once which is the optimal (find ◆ in the plot). The uniform cost algorithm finds the optimal solution with the QoS value of 510 at 61.4 seconds. On the other hand, our algorithm returns a set of solutions: 850 at 2.6 sec (the quality ratio is 510/850=0.6), 680 at 2.7 sec (the quality ratio is 510/680=0.75), 650 at 3.5 sec (the quality ratio is 510/650=0.78), 560 at 10.1 sec (the quality ratio is 510/560=0.91), 530 at 25.0 sec (the quality ratio is 510/530=0.96), and 510 at 57.7 sec (the quality ratio is 510/510=1). Then, our algorithm finally concludes that the solution is the optimal at 90.0 sec. Remark that although our algorithm has already found a solution which is the optimal at 57.7 sec, it needs more time to confirm that it is indeed optimal. Even if our any-

Table 5.2. Experiment result with different threshold: Optimal, basic anytime and dynamic anytime algorithm on QoS-aware WSC.

Problem	Timeout (sec)	Uniform cost	Basic anytime	Qual. ratio	Dynamic anytime	Qual. ratio
P1	60	Solved	Complete	1	Complete	1
P2	60	Not solved	Incomplete	0.84	Incomplete	1
P3	60	Not solved	Incomplete	0.81	Incomplete	1
P4	60	Not solved	Incomplete	0.96	Incomplete	1
P5	60	Not solved	Incomplete	0.68	Incomplete	0.81
P6	60	Not solved	Incomplete	0.85	Incomplete	0.85
P1	120	Solved	Complete	1	Complete	1
P2	120	Solved	Incomplete	1	Complete	1
P3	120	Solved	Incomplete	0.94	Incomplete	1
P4	120	Solved	Incomplete	0.96	Complete	1
P5	120	Not solved	Incomplete	0.68	Incomplete	0.81
P6	120	Not solved	Incomplete	0.85	Incomplete	0.96
P1	300	Solved	Complete	1	Complete	1
P2	300	Solved	Complete	1	Complete	1
P3	300	Solved	Complete	1	Complete	1
P4	300	Solved	Complete	1	Complete	1
P5	300	Not solved	Incomplete	0.81	Incomplete	0.87
P6	300	Not solved	Incomplete	0.96	Incomplete	0.96
P1	600	Solved	Complete	1	Complete	1
P2	600	Solved	Complete	1	Complete	1
P3	600	Solved	Complete	1	Complete	1
P4	600	Solved	Complete	1	Complete	1
P5	600	Not solved	Incomplete	0.87	Incomplete	0.87
P6	600	Solved	Incomplete	0.96	Incomplete	1

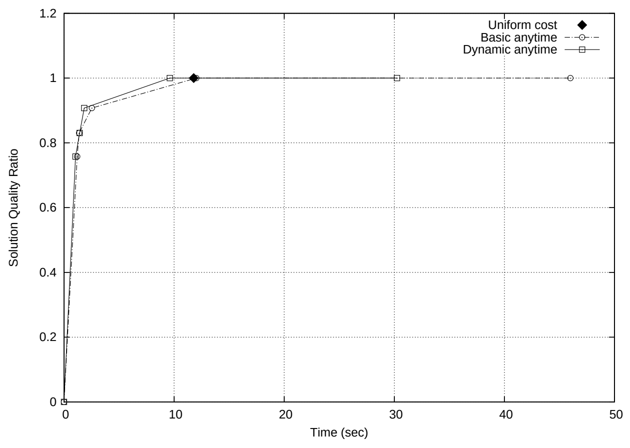
time algorithm takes longer to complete with the optimal solution, it finds several solutions with high quality (i.e., over the quality ratio of 0.9) at 10.1, 25.0, and 57.7 seconds much earlier than the uniform cost returns the solution at 61.4 sec. Even, it identifies solutions with moderate quality (i.e., between the quality ratio of 0.6 and 0.9) very quickly (i.e., at 2.6, 2.7, and 3.5 seconds). If an unlimited time is allowed, we do not need to consider using an anytime algorithm. It is, however, unreasonable assumption in a real world where service clients on web are impatient by nature. For such a case to require a fast response, anytime algorithms which return acceptable answers in an allowable time can be more satisfiable to users.

Table 5.2 presents how three search algorithms, i.e., the uniform cost search, the basic anytime algorithm based on a beam stack search, and the dynamic anytime algorithm with heuristics, solve the six QoS-aware WSC problems within specific thresholds (i.e., 60, 120, 300, 600 seconds, respectively). It shows, within each timeout, whether each algorithm solves problems and how good a so-far-best solution is. In the case of the uniform cost algorithm, since it returns the optimal answer only after finishing the necessary computation, we report only whether a given problem is solved or not. For basic and dynamic anytime algorithms, even though they can find the optimal solution, they need to visit the rest search space to confirm if the solution is optimal. 'Complete' and 'Incomplete' indicate if the algorithms accomplish even the confirmation step.

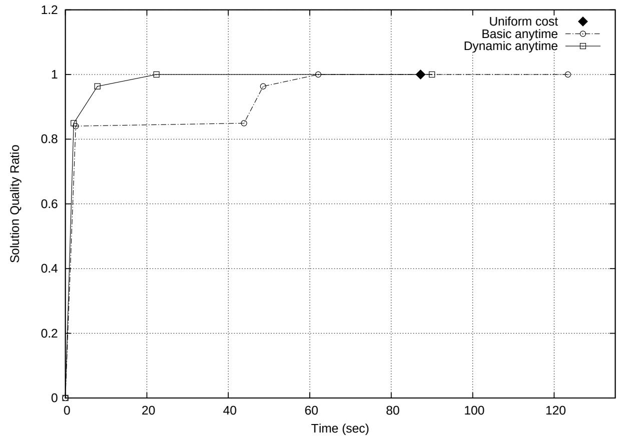
Within 60 seconds, all algorithms do not complete solving all problems except P1. However, while the optimal algorithm does not provide any answer at all, anytime algorithms generate approximate solutions of which QoS quality ratios are from 0.68 to 1. As the longer timeouts are given, the uniform cost algorithm solves more problems, i.e., it solves 1, 4, 4, and 5 problems with 60, 120, 300, and 600 seconds, respectively. Likewise, anytime algorithms complete more problems or return solutions with higher quality. The basic anytime algorithm completes 1, 1, 4, and 4 problems with 60, 120, 300, and 600 seconds, respectively, and for incomplete problems, it improves qualities of the solutions, e.g., for P5, the quality ratios of solutions returned are 0.68, 0.68, 0.81, and 0.87 with 60, 120, 300, and 600 seconds, respectively. The dynamic anytime algorithm completes 1, 3, 4, 4 problems with 60, 120, 300, 600 seconds, respectively, and for incomplete problems, it improves qualities of the solutions, e.g., for P6, the quality ratios of solutions returned are 0.85, 0.96, 0.96, and 1 with 60, 120, 300, and 600 seconds, respectively. For the problem P5, the optimal algorithm cannot solve it even in 600 seconds, which means that the algorithm does not return anything to users after a long waiting. On the other hand, even within 60 seconds, basic and dynamic anytime algorithms provide approximate solutions of which quality ratio are 0.68 and 0.81, respectively, and their quality ratios are improved with longer timeouts (up to 0.87 within 600 seconds).

In all the cases, our dynamic anytime algorithm outperforms the basic anytime algorithm; it finds higher quality solutions earlier and accomplishes the whole search faster than the basic algorithm.

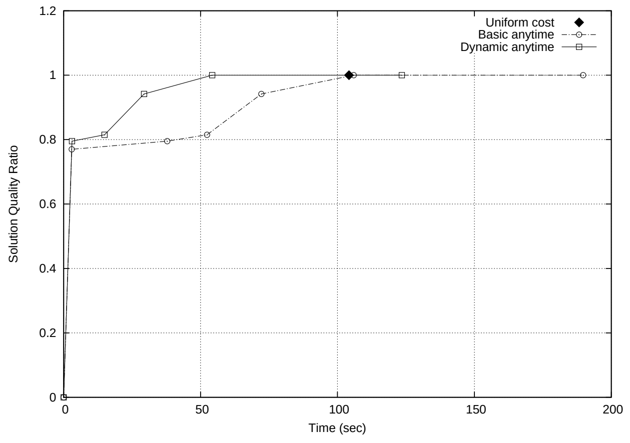
Figure 5.4 and Figure 5.5 illustrate how three algorithms work on six problems; the format is same as Figure 5.3. Anytime algorithms identify a set of approximate solutions, and the quality of their solutions is improved as time passes. In Figure 5.4 (b) and (c), and Figure 5.5 (f), the dynamic anytime algorithm notably outperforms the basic anytime algorithm, especially, at the beginning. The fact is caused by a large beam width at the early stage in the dynamic anytime algorithm. In the problem P3 (see Figure 5.4 (c)), while the basic anytime algorithm finds solutions with aggregated QoS values, 1260 and 1220, at 3.0 seconds and 37.8 seconds, respectively, the dynamic anytime algorithm first produces a solution with 1220 at 3.0 seconds, and then continues searching the better solution. Since the dynamic anytime algorithm carefully selects a branch among more candidate states at the early stage than the basic algorithm, it can bypass the solution with 1260 and find a higher quality solution with 1220 much earlier. Furthermore, since in a beam stack search, bad choices at early levels make a significant cost due to its backtracking mechanism, the ability to bypass unfavorable solutions is a valuable property of the dynamic anytime algorithm.



(a)

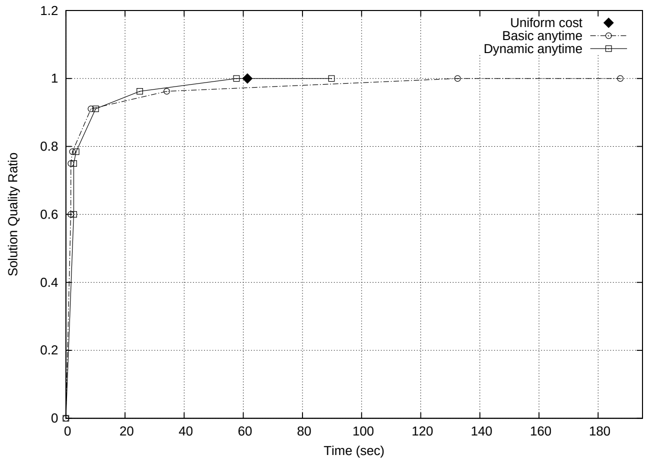


(b)

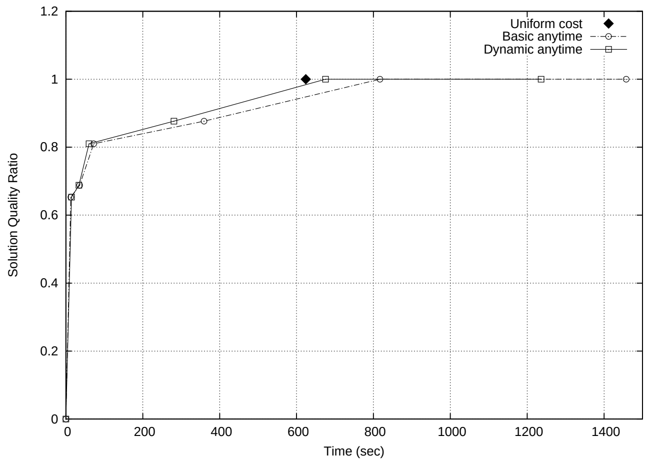


(c)

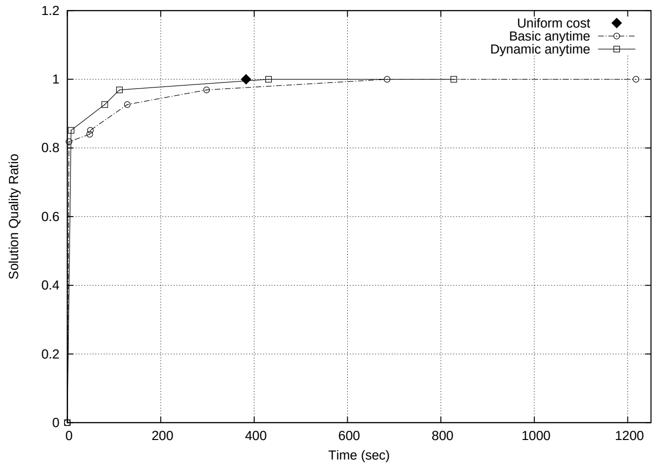
Figure 5.4. Experiment result of three algorithms (Optimal, basic anytime and dynamic anytime) on QoS-aware WSC. (a) P1, (b) P2, and (c) P3.



(d)



(e)



(f)

Figure 5.5. Continued result from Figure 5.5. (d) P4, (e) P5, and (f) P6.

Chapter 6

Conclusion

As one of vital blocks in enabling the *Semantic Web*, *web services* have received much interest from industry as well as academic research. Especially, automatic composition of web services is one of essential topics to fulfill the original goal of web services. In this dissertation, we studied this *web service composition (WSC)* problem in three different service description levels, i.e., a *signature level*, a *behavior description level*, and a *QoS description level*.

For a signature-level approach where each web service is described with its signature in *WSDL*, we first analyzed topological aspects of the real-world web services. Regardless of the matching schemes and network types, all of web service networks showed *small world* properties well and *power-law distribution* to some extent. Furthermore, as more flexible matchmaking was employed, the properties of small world and power-law network got more clearly. Since many *robust* real-world complex systems also show the similar properties of small world and power-law network, we argue that the usage of more flexible approximate matching as in *semantic* web services would make the network more robust and useful. Based on these observations, we proposed a *SAT-based* algorithm finding the shortest sequence of web services while respecting parameter types of web services. Our preliminary experiment revealed promising results where the tool found shortest sequences with logarithmic number of invocations of a SAT solver.

For a behavioral description based approach, we first formally defined a realistic model for the WSC problem on behavioral descriptions and investigated the *computational complexities* for the composition of web services on restricted (i.e., with *full* observation) and general cases (i.e., with *partial* observation). We then proved that the WSC problem with full observation is *EXP-hard* and the WSC problem with partial observation is *2-EXP-hard*. Our findings suggested that much more efforts to identify *alternative* solutions to the WSC problem be needed. As one of solutions for these high complexity problems, we proposed *approximation-based* algorithms using *abstraction* and *refinement* with two abstraction methods—*signature-preserving abstraction* and *signature-subsuming abstraction*. Our experiment on realistic problems showed promising results of our abstraction and refinement algorithms, comparing with a basic algorithm without abstraction/refinement.

For a QoS description based approach, we proposed applying *anytime algorithm* based on beam stack search to the QoS-aware WSC problem. And then, to improve the basic anytime algorithm, we proposed a *carrot-shaped dynamic beam width* with two more heuristics, *short backtracking* and *upper bound propagation*. Our experiment demonstrated that in many cases, anytime algorithms can identify composite web services with high quality much earlier than a general optimal algorithm. In addition, our heuristics can improve the quality of current approximate solutions and overall execution time.

Bibliography

[1] W3C (2007), "Web Services Description Language (WSDL) Version 2.0," `http://www.w3.org/TR/wsdl20-primer/`.

[2] W3C (2007), "SOAP Version 1.2," `http://www.w3.org/TR/soap/`.

[3] Oasis (2004), "UDDI Version 3.0.2," `http://uddi.xml.org/`.

[4] Alonso, G., F. Casati, H. Kuno, and V. Machiraju (2003) *Web Services: Concepts,Architecture, and Applications*, Springer-Verlag (ISBN: 3540440089).

[5] Medjahed, B., B. Benatallah, A. Bouguettaya, A. Ngu, and A. Elmagarmid (2003) "Business-to-Business Interactions: Issues and Enabling Technologies," VLDB Journal, **12**(1), pp. 59–85.

[6] Albert, R. and A.-L. Barabasi (2000) ""Topology of Evolving Networks: Local Events and University"," *Phys. Rev. Lett.*, **85**, pp. 5234–5237.

[7] Newman, M. E. J., S. H. Strogatz, and D. J. Watts (2001) ""Random Graphs with Arbitrary Degree Distributions and Their Applications"," *Phys.Rev. E*, **64**(2), p. 026118.

[8] Xu, L., F. Hutter, H. H. Hoos, and K. Leyton-Brown (2007) "SATzilla-07: The design and analysis of an algorithm portfolio forSAT," in *The 13th International Conference on Principles and Practice of Constraint Programming CP 2007*, pp. 712–727.

[9] Srivastava, B. and J. Koehler (2003) "Web service composition: Current solutions and open problems," in *In* ICAPS Workshop on Planning for Web Services, pp. 28–35.

[10] Sirin, E., B. Parsia, D. Wu, J. Hendler, and D. Nau (2004) "HTN planning for web service composition using SHOP2," Journal of Web Semantics, **1**(4), pp. 377–396.

[11] Oasis (2007), "Web services business process execution language version 2.0," `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`.

[12] W3C (2004), "OWL-S: Semantic markup for web services," `http://www.w3.org/Submission/OWL-S/`.

[13] W3C (2006), "Web Services Policy (WS-Policy) Version 1.2," `http://www.w3.org/Submission/WS-Policy/`.

[14] IBM (2004), "WSLA (Web Service Level Agreements) project," `http://www.research.ibm.com/wsla/documents.html`.

[15] OGF (2007), "Web Services Agreement Specification (WS-Agreement)," `http://www.ogf.org/documents/GFD.107.pdf`.

[16] Zilberstein, S. (1996) "Using Anytime Algorithms in Intelligent Systems," *AI Magazine*, **17**(3), pp. 73–83.

[17] Zhou, R. and E. A. Hansen (2005) "Beam-Stack Search: Integrating Backtracking with Beam Search," in *ICAPS*, pp. 90–98.

[18] "The Web Service Challenge," `http://ws-challenge.org/`.

[19] Berners-Lee, T., J. Hendler, and O. Lassila (2001) "The semantic web," Scientic American, **284**(5), pp. 34–43.

[20] Casati, F. and M. Shan (2001) "Models and languages for describing and discovering e-services," in *Proceedings of ACM Sigmod International Conference on Management of Data*, p. 626.

[21] Tsur, S., S. Abiteboul, R. Agrawal, U. Dayal, J. Klein, and G. Weikum (2001) "Are Web services the next revolution in e-commerce?" in *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 614–617.

[22] W3C (2002), "W3C Web Services Activity," `http://www.w3c.org/2002/ws/`.

[23] Medjahed, B. (2004) *Semantic Web Enabled Composition of Web Services*, Ph.D. thesis, Virginia Tech.

[24] xMethod "http://www.xmethod.com," .

[25] Bindingpoint "http://www.bindingpoint.com (Last accessed May 1, 2006)," .

[26] RemoteMethods "http://www.RemoteMethods.com," .

[27] eSynaps "http://www.robtex.com/dns/esynaps.com.html," .

[28] Schlosser, M. T., M. Sintek, S. Decker, and W. Nejdl (2002) "A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services," in *Peer-to-Peer Computing*, pp. 104–111.

[29] "Speed-R: Semantic P2P environment for diverse Web Service registries," http://webster.cs.uga.edu/mulye/SemEnt/Speed-R.html.

[30] Haas, H. (2004) *Architecture and future of web services: from SOAP to semantic web services.*

[31] Hull, R. (2003) "E-Service Composition: Models and Formalisms," in *Description Logics.*

[32] Berardi, D., D. Calvanese, D. Giacomo, Giuseppe, M. Lenzerini, and M. Mecella (2005) "Automatic Service Composition Based on Behavioral Descriptions," *Int. J. of Cooperative Information Systems*, **14**(4), pp. 333–376.

[33] Semantic Web (2010), "Ontology," `http://semanticweb.org/wiki/Ontology`.

[34] Srivastava, B. and J. Koehler (2003) "Web Service Composition: current solutions and open problems," in *In ICAPS workshop on Planning for Web Services.*

[35] Casati, F., S. Ilnicki, and L. Jin (2000) "Adaptive and dynamic service composition in EFlow," in *Proceedings of 12th International Conference on Adavanced Information Systems Engineering (CAiSE).*

[36] Schuster, H., D. Georgakopoulos, A. Cichocki, and D. Baker (2000) "Modeling and composing service-based and reference process-based multi-enterprise process," in *Proceedings of 12th International Conference on Adavanced Information Systems Engineering (CAiSE).*

[37] Paolucci, M., T. Kawamura, T. R. Payne, and K. P. Sycara (2002) "Semantic Matching of Web Services Capabilities," in *International Semantic Web Conference*, pp. 333–347.

[38] Mecella, M., B. Pernici, and P. Craca (2001) "Compatibility of e -Services in a Cooperative Multi-platform Environment," in *TES*, pp. 44–57.

[39] Wu, J. and Z. Wu (2005) ""Similarity-based Web Service Matchmaking"," in *IEEE Int'l Conf. on Services Computing (SCC)*, Orlando, FL, USA.

[40] Wang, Y. and E. Stroulia (2003) ""Semantic Structure Matching for Assessing Web Service Similarity"," in *IEEE Int'l Conf. on Services Computing (SCC)*, Trento, Italy.

[41] Dong, X., A. Halevy, J. Madhavan, E. Nemes, and J. Zhang (2004) "“Similarity Search for Web Services”," in *VLDB*, Toronto, Ontario, Canada.

[42] Oh, S.-C., D. Lee, and S. Kumara (2008) "Effective Web Service Composition in Diverse and Large-Scale Service Networks," *IEEE Trans. on Services Computing (TSC)*, **1**(1), pp. 15–32.

[43] Bilenko, M., W. Cohen, S. Fienberg, J. Mooney, and R. Ravikumar (2003) "Adaptive name matching in information integration," *IEEE Intelligent Systems*, **18**(5), pp. 16–23.

[44] Fellbaum (Eds), C. (1998) *"WordNet: An Electronic Lexical Database"*, The MIT Press.

[45] Medjahed, B., A. Bouguettaya, and A. K. Elmagarmid (2003) "Composing Web services on the Semantic Web," *VLDB J.*, **12**(4), pp. 333–351.

[46] Sirin, E., J. A. Hendler, and B. Parsia (2003) "Semi-automatic Composition ofWeb Services using Semantic Descriptions," in *WSMAI*, pp. 17–24.

[47] Cardoso, J. and A. P. Sheth (2003) "Semantic E-Workflow Composition," *J. Intell. Inf. Syst.*, **21**(3), pp. 191–225.

[48] Cardoso, J. and A. P. Sheth (2003) "Semantic Web Processes: Semantics Enabled Annotation, Discovery, Composition and Orchestration of Web Scale Processes," in *WISE*, pp. 375–376.

[49] Y. Kalfoglou, W. M. Schorlemmer, A. P.Sheth, S. Staab, and M. Uschold (2005) "Semantic Interoperability and Integration," in *Semantic Interoperability and Integration,* vol. 04391 of *Dagstuhl Seminar Proceedings*, IBFI, Schloss Dagstuhl, Germany.

[50] Sivashanmugam, K., K. Verma, A. P. Sheth, and J. A. Miller (2003) "Framework for Semantic Web Process Composition," *Special Issue of the International Journal of Electronic Commerce.*

[51] Akkiraju, R., B. Srivastava, A.-A. Ivan, R. Goodwin, and T. F. Syeda-Mahmood (2006) "SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition," in *AAAI.*

[52] Mrissa, M., C. Ghedira, D. Benslimane, Z. Maamar, F. Rosenberg, and S. Dustdar (2007) "A context-based mediation approach to compose semantic Web services," *ACM Transactions on Internet Technology*, **8**(1), p. 4.

[53] Sycara, K. P., M. Klusch, S. Widoff, and J. Lu (1999) "Dynamic Service Matchmaking Among Agents in Open Information Environments," *SIGMOD Record*, **28**(1), pp. 47–53.

[54] Ankolekar, A., M. H. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, K. P. Sycara, and H. Zeng (2001) "DAML-S: Semantic Markup for Web Services," in *The Emerging Semantic Web*.

[55] Maedche, A. and S. Staab (2002) ""Measuring Similarity between Ontologies"," in *European Conf. on Knowledge Acquisition and Management (EKAW)*, Siguenza, Spain.

[56] Li, L. and I. Horrocks (2003) "A software framework for matchmaking based on semantic web technology," in *WWW*, pp. 331–339.

[57] Zeng, L., B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng (2003) "Quality driven web services composition," in *WWW*, pp. 411–421.

[58] Zeng, L., B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang (2004) "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Software Eng.*, **30**(5), pp. 311–327.

[59] Aggarwal, R., K. Verma, J. A. Miller, and W. Milnor (2004) "Constraint Driven Web Service Composition in METEOR-S," in *IEEE SCC*, pp. 23–30.

[60] Ardagna, D. and B. Pernici (2007) "Adaptive Service Composition in Flexible Processes," *IEEE Trans. Software Eng.*, **33**(6), pp. 369–384.

[61] Ardagna, D. and B. Pernici (2005) "Global and Local QoS Constraints Guarantee in Web Service Selection," in *ICWS*, pp. 805–806.

[62] Yu, T., Y. Zhang, and K.-J. Lin (2007) "Efficient algorithms for Web services selection with end-to-end QoS constraints," *ACM Trans. Web*, **1**(1), p. 6.

[63] Oldham, N., C. Thomas, A. P. Sheth, and K. Verma (2004) "METEOR-S Web Service Annotation Framework with Machine Learning Classification," in *SWSWPC*, pp. 137–146.

[64] Grefen, P. W. P. J., K. Aberer, H. Ludwig, and Y. Hoffner (2001) "CrossFlow: Cross-Organizational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises," *IEEE Data Eng. Bull.*, **24**(1), pp. 52–57.

[65] Alrifai, M. and T. Risse (2009) "Combining global optimization with local selection for efficient QoS-aware service composition," in *WWW*, pp. 881–890.

[66] Canfora, G., M. D. Penta, R. Esposito, and M. L. Villani (2005) "An approach for QoS-aware service composition based on genetic algorithms," in *GECCO*, pp. 1069–1075.

[67] Dayal, U., K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y.-K. Kim (eds.) (2006) *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, ACM.

[68] Yu, Q. and A. Bouguettaya (2008) "Framework for Web service query algebra and optimization," *ACM Trans. Web*, **2**(1), pp. 1–35.

[69] Berardi, D., D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella (2005) "Automatic Composition of Web Services in Colombo," in *SEBD*, pp. 8–15.

[70] Berardi, D., D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella (2005) "Automatic Composition of Transition-based Semantic Web Services with Messaging," in *VLDB*, pp. 613–624.

[71] McIlraith, S. A. and T. C. Son (2002) "Adapting Golog for Composition of Semantic Web Services," in *KR*, pp. 482–496.

[72] Giacomo, G. D., Y. Lespérance, and H. J. Levesque (2000) "ConGolog, a concurrent programming language based on the situation calculus," *Artif. Intell.*, **121**(1-2), pp. 109–169.

[73] Narayanan, S. and S. A. McIlraith (2002) "Simulation, verification and automated composition of web services," in *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pp. 77–88.

[74] Hull, R. and J. Su (2005) "Tools for composite web services: a short overview," *SIGMOD Record*, **34**(2), pp. 86–95.

[75] Pistore, M., P. Traverso, P. Bertoli, and A. Marconi (2005) "Automated synthesis of executable web service compositions from BPEL4WS processes," in *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pp. 1186–1187.

[76] Pistore, M., P. Traverso, and P. Bertoli (2005) "Automated composition of web services by planning in asynchronous domains," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pp. 2–11.

[77] Bertoli, P., M. Pistore, and P. Traverso (2006) "Automated web service composition by on-the-fly belief space search," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pp. 358–361.

[78] Fan, W., F. Geerts, W. Gelade, F. Neven, and A. Poggi (2008) "Complexity and composition of synthesized web services," in *PODS '08: Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 231–240.

[79] Brogi, A. (2010) "On the Potential Advantages of Exploiting Behavioural Information for Contract-based Service Discovery and Composition," *Journal of Logic and Algebraic Programming*, pp. 1–10.

[80] Reif, J. (1984) "The complexity of two-player games of incomplete information," *Journal on Computer and System Sciences*, **29**, pp. 274–301.

[81] Thomas, W. (2002) "Infinite games and verification," in *Proceedings of the International Conference on Computer Aided Verification (CAV'02)*, LNCS 2404, Springer, pp. 58–64.

[82] Antoniotti, M. (1995) *Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the control-D system*, Ph.D. thesis, New York University.

[83] Pnueli, A. and R. Rosner (1989) "On the synthesis of a reactive module," in *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1989)*, pp. 179–190.

[84] Vardi, M. (1995) "An automata-theoretic approach to fair realizability and synthesis," in *Proceedings of International Conference on Computer Aided Verification (CAV'95)*, pp. 267–278.

[85] Kupferman, O. and M. Vardi (1997) "Synthesis with incomplete information," in *Proceedings of the 2nd International Conference on Temporal Logic*, pp. 91–106.

[86] Tsitsiklis, J. N. (1989) "On the Control of Discrete Event Dynamical Systems," *Mathematics of Control, Signals and Systems*, **2**(2), pp. 95–107.

[87] Rudie, K. and J. C. Willems (1995) "The Computational Complexity of Decentralized Discrete-Event Control Problems," *IEEE Transactions on Automatic Control*, **40**(7), pp. 1313–1319.

[88] Herzig, A., J. Lang, D. Longin, and T. Polacsek (2000) "A logic for planning under partial observability," in *Proceedings of National Conference on Artificial Intelligence (AAAI-00)*, pp. 768–773.

[89] Bertoli, P. and M. Pistore (2004) "Planning with Extended Goals and Partial Observability," in *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pp. 270–278.

[90] Rintanen, J. (2004) "Complexity of planning with partial observability," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pp. 345–354.

[91] Moffitt, M. (2007) "On the partial observability of temporal uncertainty," in *Proceedings of National Conference on Artificial Intelligence (AAAI-07)*, pp. 1031–1037.

[92] Pnueli, A. (1981) "The Temporal Semantics of Concurrent Programs." *Theoretical Computer Science*, **13**, pp. 45–60.

[93] Emerson, E. (1990) "Temporal and modal logic," in *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), vol. B, Elsevier Science Publishers, pp. 995–1072.

[94] Huang, W., Z. Wen, Y. Jiang, and L. Wu (2007) "Observation reduction for strong plans," in *Proceedings of the 23th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 1930–1935.

[95] Armano, G., G. Cherchi, and E. Vargiu (2003) "A parametric hierarchical planner for experimenting abstraction techniques," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pp. 936–941.

[96] Smith, T., D. R. Thompson, and D. Wettergreen (2007) "Generating exponentially smaller pomdp models using conditionally irrelevant variable abstraction," in *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS 2007)*, pp. 304–311.

[97] Watts, D. J. and S. H. Strogatz (1998) ""Collective Dynamics of 'Small-World' Networks"," *Nature*, **393**(4), pp. 440–442.

[98] Albert, R., H. Jeong, and A.-L. Barabasi (1999) ""The Diameter of the World Wide Web"," *Nature*, **401**, pp. 130–131.

[99] Denning, P. J. (2004) ""Network Laws"," *Comm. ACM*, **47**(11), pp. 15–20.

[100] Blake, M. B., W. K.-W. Cheung, M. C. Jaeger, and A. Wombacher (2007) "WSC-07: Evolving the web services challenge," in *CEC/EEE*, pp. 505–508.

[101] Milgram, S. (1967) "The Small World Problem," *Psychology Today*, **1**(6), pp. 61–67.

[102] Watts, D. J. (1999) *"The Dynamics of Networks between Order and Radnomness"*, Princeton Univ. Press, Princeton, NJ.

[103] RDF-Schema http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[104] WSDL-S http://lsdis.cs.uga.edu/library/download/wsdl-s.pdf.

[105] Baeza-Yates, R. and B. Ribeiro-Neto (1999) *"Modern Information Retrieval"*, Addison–Wesley.

[106] Fan, J. and S. Kambhampati (2005) ""A Snapshot of Public Web Services"," *SIGMOD Record*, **34**(1), pp. 24–32.

[107] Newman, M. E. J. (2005) ""Power Laws, Pareto Distributions and Zipf's Law"," *Contemporary Physics*, **46**, pp. 323–351.

[108] Kil, H., S.-C. Oh, and D. Lee (2006) ""On the topological landscape of web services matchmaking"," in *In VLDB Workshop on Semantic Matchmaking and Resource Retrieval (SMR)*.

[109] "The International SAT Competitions," `http://www.satcompetition.org/`.

[110] Narayanan, S. and S. McIlraith (2003) "Analysis and simulation of web services," *Computer Networks*, **42**(5), pp. 675–693.

[111] Balcázar, J. L., J. Díaz, and J. Gabarró (1988) *Structural complexity 1*, Springer-Verlag.

[112] Balcázar, J. L., J. Díaz, and J. Gabarró (1990) *Structural complexity 2*, Springer-Verlag.

[113] Papadimitriou, C. M. (1994) *Computational complexity*, Addison-Wesley.

[114] Chandra, A., D. Kozen, and L. Stockmeyer (1981) "Alternation," *Journal of the ACM*, **28**(1), pp. 114–133.

[115] Traverso, P. and M. Pistore (2004) "Automated composition of semantic web services into executable processes," in *Proceedings of International Semantic Web Conference (ISWC'04)*, pp. 380–394.

[116] Bertoli, P., A. Cimatti, M. Roveri, and P. Traverso (2002) "Strong planning under partial observability," Artificial Intelligence, **170**(4), pp. 337–384.

[117] Barbon, F., P. Traverso, M. Pistore, and M. Trainotti (2006) "Run-time monitoring of instances and classes of web service compositions," in *In* ICWS, pp. 63–71.

[118] Bellman, R. (1957) *Dynamic Programming*, Princeton University.

[119] Zhou, R. and E. A. Hansen (2004) "Breadth-First Heuristic Search," in *ICAPS*, pp. 92–100.

[120] Russell, S. and P. Norvig (2003) *Artificial Intelligence: A Modern Approach*, 2nd ed., Prentice-Hall.

Vita

Hyunyoung Kil

Hyunyoung Kil was born in Seoul, Republic of Korea in 1975. She received her B.S. degree and M.S. degree from the Department of Computer Science and Engineering, Korea University, Seoul, Korea, in 1998 and in 2001, respectively. Then, she received her M.S. degree from the Department of Computer Science, University of Pennsylvania, Philadelphia, PA, USA in 2003. Then, she joined Ph.D program in the Department of Computer Science and Engineering at the Pennsylvania State University, State College, PA, USA in 2004. Her current research focuses on Automatic web service composition, automated planning and controller synthesis, and digital library.