The Pennsylvania State University
The Graduate School

EFFECTIVE WEB-SERVICE COMPOSITION IN DIVERSE AND LARGE-SCALE SERVICE NETWORKS

A Thesis in
Industrial Engineering
by
Seog-Chan Oh

© 2006 Seog-Chan Oh

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2006

The thesis of Seog-Chan Oh was reviewed and approved* by the following:

Soundar R.T. Kumara
Distinguished Professor of Industrial Engineering
Thesis Co-Advisor, Chair of Committee

Dongwon Lee
Assistant Professor of College of Information Sciences and Technology
Thesis Co-Advisor

Tom M. Cavalier
Professor of Industrial Engineering

Timothy W. Simpson
Professor of Mechanical and Industrial Engineering

Ling Rothrock
Assistant Professor of Industrial Engineering

Richard J. Koubek
Professor of Industrial Engineering
Head of the Harold and Inge Marcus Department of Industrial and Manufacturing Engineering

*Signatures are on file in the Graduate School.

Abstract

Web services are considered to be a potential *silver bullet* for the envisioned *Service Oriented Architecture*, in which loosely coupled software components are published, located and executed as parts of distributed applications. Web services intend to take the public web and today's distributed systems to unexplored efficiencies while suggesting flexible interfaces for promoting a wide spectrum of activities in tomorrow's service networks. The main research focus of web services is to achieve interoperability between distributed and heterogeneous applications. Therefore, flexible composition of web services to fulfill the requirements of tasks is one of the most important objectives in this research field. Applications including B2B E-commerce and E-government as well as in the public web, are expected to benefit greatly from web service composition.

Until now, service composition has been an impromptu, tedious, and fallible process involving continuous low-level programming. Furthermore, as the number of available web services increases, finding the right web services to fulfill the given goal becomes intractable. In this dissertation, we propose an AI planning-based framework for the automatic composition of web services. For this purpose, we explore the following issues. First, we formulate the web-service composition problem in terms of AI planning and network optimization problems to investigate its complexity. Second, we analyze publicly available web service sets using complex network analysis techniques. Third, we develop a novel web-service benchmark called *WSBen*. Fourth, we develop a novel AI planning-based heuristic web-service composition algorithm named *WSPR*. Finally, we conduct extensive experiments to verify WSPR against state-of-the-art AI planners. It is our hope that WSPR and WSBen will provide useful insights for researchers to develop web-service discovery and composition algorithms, and software.

Table of Contents

List of Figures

List of Tables

List of Symbols

W	Set of web services, $w \in W$, p. 99
w	Web service, p. 12
w^i	Set of input parameters of w, p. 12
w^o	Set of output parameters of w, p. 12
P	Set of parameters, $p \in P$, p. 12
r	Request of service discovery or composition, p. 12
r^i	Set of initial parameters of r, p. 12
r^o	Set of goal parameters of r, p. 12
Π	Propositional STRIPS model consisting of 4-tuple, p. 12
Ψ	State space model corresponding to Π, p. 13
S	Set of states, $s \in S$ are collection of parameters in P, p. 13
s_0	Initial state $s_0 \in S$ is such that $s_0 = r^i$, p. 13
S_G	Goal states $s \in S_G$ are such that $r^o \subseteq s$, p. 13
$\Omega(s)$	Set of web services $w \in W$ such that $w^i \in s$, p. 13
f	Transition function $f(w, s) = s'$ that maps a state s into a state s' such that $s' = s \cup w^o$ for $w \in \Omega(s)$, p. 13

$c(w)$	Invocation cost of w, p. 13
$G_s(V_s, E_s)$	State node network, p. 15
b_i	Amount of flow that enters the network at node $s_i \in V_s$, $i = 0, \ldots, n$, p. 16
$fl_{i,j}$	Amount of flow which is either 0 or 1 on arc $(s_i, s_j) \in E_s$, p. 16
$c_{i,j}$	Cost per unit flow on arc $(s_i, s_j) \in E_s$. $c_{i,j}$ is $c(w)$ when $s_j = f(w, s_i)$, p. 16
λ_i,λ^*	λ_i is the cost of the path from s_i to s_n. λ_i^* is the minimum cost of λ_i. λ_i^*s are also called *labels*, p. 17
U	Set of variables used in an instance of 3SAT, p. 19
$T(k)$,$F(k)$,$V(k)$	Propositional conditions (parameters) associated with $u_k \in U$, p. 19
$C(j)$	Propositional condition (parameter) to represent whether the jth clause of an instance of 3SAT is satisfied or not, p. 19
wt_k,wf_k,wv_{k1},wv_{k2}	Four web services associated with $u_k \in U$, p. 19
$Keep(A)$	Maintenance action of an action A in Graphplan. It is also called the no-op of A, p. 33
$X_{e,i}$	Decision variable such that if effect e is true in period i of Graphplan, then 1, otherwise 0, p. 34
$Y_{a,i}$	Decision variable such that if action a is carried out in period i of Graphplan, then 1, otherwise 0, p. 34
$Y_{e,i}$	Decision variable to represent a maintenance action for effect e of Graphplan during period i, p. 34
$G_p(V_p, E_p)$	Parameter node network, p. 37
$G_{op}(V_{op}, E_{op})$	Operation node network, p. 37
$G_{ws}(V_{ws}, E_{ws})$	Web-service node network, p. 37
$G_{op}^f(V_{op}^f, E_{op}^f)$	Full-matching operation node network, p. 38

$\#(p)$ Parameter usage defined as the frequency of the parameter p in the web service repository W, p. 38

$R_{(N,k)}$ Regular network, p. 43

L Average shortest distance between reachable pairs of vertices in a network. $L(p)$ is defined as L of the randomly rewired Watts-Strogatz graph with probability p. L_{random} is identical to $L(1)$, p. 44

C Average clustering coefficient. $C(p)$ is defined as C of the randomly rewired Watts-Strogatz graph with probability p. C_{random} is identical to $C(1)$, p. 44

$P_w(v)$ $P_w(v) \propto v^{(-\gamma)}$. It is used to approximate the number of nodes that have v number of neighbor nodes, p. 45

$\#In(p)$ Number of web services that contain a parameter p in their input parameter, p. 59

$\#Out(p)$ Number of web services that contain a parameter p in their output parameter, p. 59

$Pr(p_2|p_1)$ Co-occurrence probability, where p_1 and p_2 are parameters, p. 59

$G_{cl}(V_{cl}, E_{cl})$ Parameter cluster network, p. 59

xTS, yTS Two input parameter frameworks of WSBen, p. 60

J Set of clusters, p. 60

G_r Graph model used to specify the underlying topology of a parameter cluster network, p. 60

η Parameter condense rate, p. 60

M_p Minimum number of parameters a cluster can contain, p. 60

Pa_j Set of parameters contained in cluster j, p. 63

$baTS$ Test set framework instanced from xTS.
$baTS = \langle$ 100,Barabasi-Albert(100,6),0.8,5,$|W|$ $\rangle$, p. 65

$nwsTS$ Test set framework instanced from xTS.
$nwsTS = \langle$100,Newman-Watts-Strogatz(100,6,0.1),0.8,5,$|W|\rangle$, p. 65

$erTS$ Test set framework instanced from xTS.
$erTS = \langle$100,Erdos-Renyi(100,0.06),0.8,5,$|W|\rangle$, p. 65

$g_{r^i}(p)$ Cost of achieving $p \in P$ from a state r^i. The forward searching step of WSPR obtains $g_{r^i}(p)$ for all $p \in P$, p. 82

$Ow(p)$ Set of web services: $Ow(p) = \{w \in W | p \in w^o\}$, p. 83

$PD_{ws}(p)$ Inverted index that contains the set of predecessor web services of p. It is used for the regression search of WSPR, p. 83

$subGoal$ Set of parameters that are required to be searched. *subGoal* is composed in the intermediate process of the regression search of WSPR, p. 84

$wSpace$ Set of web services $w \in W$, such that $w_i \in PD_{ws}(p)$, where $p \in subGoal$, p. 84

$h_{sg}(w)$ Heuristics used for selecting a web service in the regression searching process of WSPR, p. 84

$soln$ Sequence of web services that is a solution to r, p. 89

$Time$ Performance measure to see how long an algorithm takes to find a solution in milliseconds, p. 99

$\#W$ Performance measure to see the number of web services in a solution, p. 99

$Co(j)$ Co-occurrence probability of parameters in a cluster $j \in J$, p. 153

$Rl(j)$ Association distribution of a cluster $j \in J$, p. 154

uTS Test set framework instanced from yTS.
$uTS = \langle 100, uni(0.2), 1, uni(0.2), |W| \rangle$, p. 155

bTS Test set framework instanced from yTS.
$bTS = \langle 100, bell(0.3, 0.5), 1, bell(0.3, 0.5), |W| \rangle$, p. 155

sTS Test set framework instanced from yTS.
$sTS = \langle 100, skew(0.5, 0.5), 1, skew(0.5, 0.5), |W| \rangle$, p. 155

Acknowledgments

Writing this dissertation marks the happy conclusion of the journey that I started many years ago. Throughout the journey, I greatly and sincerely benefited from the support and companionship of many people.

First and foremost, I would like to express my gratitude to Dr. Soundar Kumara, Distinguished Professor of Industrial Engineering, the Pennsylvania State University for his support and encouragement throughout my Ph.D study. He is an enthusiastic researcher who is always looking for novel perspectives and motivating his students toward innovations. I have tried to learn his insightful perspectives for the last four years since I started working with him. I would also like to thank my co-advisor, Dr. Dongwon Lee, for providing the unmeasurable guidance and feedback on the research problems and issues studied in this dissertation. As I retrospect, I realize that I have learned so many things from him, technical or non-technical, directly or indirectly, that practically shaped who I am now as a scholar. I would also like to thank the other committee members, Dr. Tom Cavalier, Dr. Timothy Simpson, and Dr. Ling Rothrock, for their guidance and suggestions during my research.

I would specially like to thank to my senior friend, Yunho Hong, and his wife, Hyeyim Na, have given me meaningful advice and invaluable support whenever I was in trouble. In finishing my Ph.D study, I would like to give thanks to many colleagues in the LISQ research group, whom I have spent many hours together for last four years; Seokcheon Lee, Changsu Ok, Seungki Moon, Jindae Kim, Hari Prasad, Nathan Gnanasambandam, Usha Nandini Raghavan, Chunglin Chang, Yiyu Chen, and Christopher Carrino. Thanks to the students, I was actually able to enjoy being in the office at the Leonhard Building. I have always admired Hari's clean logical thinking and sharp insights. Seungki and I had much fun together in playing raquetball, eating snacks, and chatting over coffee.

During my Ph.D study, I was fortunate to have chances to work closely with

many bright students in the PIKE research group: Hyunyoung Kil, Byungwon On, Ergin Elmacioglu, and Eric Larson. They all contributed to my dissertation one way or the other. Without Hyunyoung's unmeasurable support, I could not have finished my dissertation. Without Byungwon's generous encouragement, I could not have finished my work. Thanks to Ergin's support of his deep knowledge, I was able to finish my experiments. I am indebted to Eric with whom I attended the EEE05 web service contest that later became the basis for me to win the first runner-up award in the ICEBE05 web service contest.

I especially like to thank my seniors who graduated from IE department, Penn State: Dr. Shangtae Yee at General Motors, Dr. Taioun Kim at Kyungseong University, Dr. Younghan Lee at Dongguk University, and Dr. Cheunghwa Lee, although they would never realize how much influential they were to me.

My dear friends, Kwangho Shin, Seongmo Kim, Dr. Youngbong Jang, Dr. Seonghwan Min, Dr. Kwangku Seo, helped me to maintain sanity during the Ph.D program by way of occasional emails and phone calls. My senior, Jeongwok Kim, Director of the consulting division at Accenture in South Korea, has always been tremendous comforts to me, for which I deeply appreciate.

Sincerely, I would like to thank my mother and brothers and sisters, from the bottom of my heart, for their prayers and confidence in me. I am really proud of them all. Finally, I would like to express that this dissertation is dedicated to my late father who will always be immortal in my heart.

Finally, I would like to acknowledge the partial support provided by DARPA Grant #:MDA972-01-1- 0038, NSF Grant #:CMMI-0537992. The findings and reporting are my own conclusion and does not reflect the positions of the above funding agencies.

Chapter 1

Introduction

Web services are often considered to be one of the most important and vital building blocks for the "Semantic Web" [15]. As such, the industrial support for web services has grown dramatically in recent years. For example, it is expected that by 2007, 72% of all application development software will support web services, and 45% of all types of software will be web-services enabled [25]. As a growing number of web services are available on the web and in organizations, finding and composing the right set of web services becomes ever more important.

The main research focus of web services is to achieve interoperability between distributed and heterogeneous applications. Therefore, flexible composition of web services in order to fulfill the requirements of the tasks is one of the most important objectives in this research field. To date, however, enabling composite services has largely been an ad hoc, time-consuming, and error-prone process involving repetitive low-level programming. As a result, in recent years, a plethora of research and products on web service composition problems have appeared[1]. In addition, the web-services research community has hosted competition programs (e.g., EEE05 [37] and ICEBE05 [54]) to solicit algorithms and software to discover pertinent web services and compose them to make value-added functionality. However, little research has addressed the composition problem when the number of web services becomes very large and the underlying web-service network topologies are diverse. The current work is differentiated by the fact that it improves the

[1] At the time this thesis is written (May 1, 2006), there are about 900 and 80 scholarly articles mentioning "web-services composition" at Google Scholar and CiteSeer, respectively.

previous AI planning methods with a novel heuristic and develops test sets based on the characteristics of real web services.

1.1 Introduction to Web Services

A web service [114] is a piece of XML-based software interface that can be invoked over the Internet and can be roughly viewed as a next-generation successor of the Common Object Request Broker Architecture (CORBA) [29] or the Remote Procedure Call (RPC) [52] technique. The main benefits of web services are as follows:

- Inter-operability: the XML standard allows applications on any platform to communicate with other web-service applications. While other factors may change, the web services interface remains accessible.
- Ease of use: as long as developers adhere to web-service standards, they are free to use their own programming language, architecture, and implementation strategy.
- Reusability : web services are component-based so that they allow interfaces with potentially unlimited sources.
- Ubiquitous computing: web services are accessible everywhere because they use the Internet. An added advantage is that they have been developed to comply with existing Internet-based security measures, such as firewalls.

The web-service architecture is constructed so that web services allow any piece of software to communicate with a standard XML messaging system. For implementing the XML-based communication, the following XML-based technologies are necessary:

- WSDL: Web Services Description Language (WSDL) [115] is an XML-based format for specifying the interface to a web service. The WSDL details the service's available operations and parameter types, as well as the actual SOAP endpoint for the service. In essence, WSDL is the "user's manual" for web services.

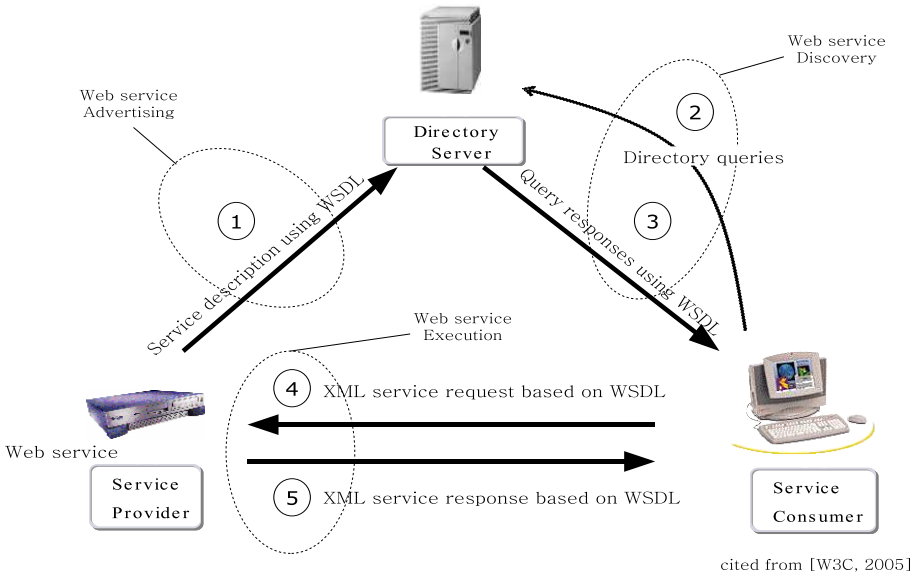


Figure 1.1: How web services work

- SOAP: Simple Object Access Protocol (SOAP) [112] is an XML-based protocol for exchanging information in a decentralized and distributed environment. It defines a mechanism to exchange commands and parameters between clients and servers. Like web services as a whole, SOAP is independent of the platform, object model, and programming language being used.

- UDDI: Universal Description, Discovery and Integration (UDDI) [113] is the meeting place for web services. The UDDI registry stores descriptions about companies and the services they offer in a common XML format. As such, the UDDI acts effectively a "yellow pages" for web services.

In Figure 1.1, we explain how web services work. Typically, it starts with a `Service Provider`, which is a web service.

1. First, the `Service Provider` registers its detailed service specification (a WSDL file) to a `Directory Server`, which has the role of the yellow pages server. This process corresponds to ① in Figure 1.1.

2. Second, a `Service Consumer` (e.g., software agent, program, or human) then finds the `Service Provider` that can satisfy certain needs from the `Directory Server` by using UDDI protocol. The WSDL file containing a

detailed specification about the `Service Provider` is obtained. This process corresponds to ② and ③ in Figure 1.1.

3. Third, using the known API and data types specified in the WSDL file, the `Service Consumer` sends a request to the `Service Provider` via a standard message protocol, SOAP, and in return, receives a response from the `Service Provider`. This process corresponds to ④ and ⑤ in Figure 1.1.

Unlike conventional programming interfaces, web services are self-explanatory. Specifically, by interpreting XML tags, applications can interpret the meanings of operations and data in an easier way than before. Therefore, locating the correct services and combining them to form more complex services becomes an increasingly important task on the web. It is evident that when there are a large number of web-service based agents available, it is essential to quickly discover and combine web services to satisfy the given request. Recently, composition of services has received much interest due to its potential to tackle many adaptive system architecture issues that were previously hard to overcome by other computing paradigms. Some of the issues related to the service composition are listed in the following [73, 129]:

- Most service domains are normally large in size and service offerings are dynamic, with new services becoming available on a daily basis.

- There are usually multiple services that offer seemingly similar features but have some variation in details (e.g., different parameters for invocation, different access interfaces, different costs, different quality).

- Composition of services needs to be generated on demand by the requests of customers. Customer requirements, such as expending of business transactions, duration of service invocation, and different preferences, are very important in a service composition. It is a complex issue to deal with, since there are complex and often conflicting relationships among these requirement factors.

In the following section, an explanation of web-service composition is carried out with an example.

1.2 Motivating Example

In web services enabled networks, typically a client program first locates a web services server that can satisfy certain requests from a yellow page (UDDI), and obtain a detailed specification (WSDL) about the service. Then, using the known API in the specification, the client sends a request to the web service considered via a standard message protocol (SOAP), and in return, it receives a response from the service. As mentioned before, web services are self-explanatory; by interpreting XML tags, applications can understand the semantics of operations. In particular, a problem of practical interest concerns the following two issues. Given a request r, among thousands of candidate web services found in UDDI: (1) How to find matching services that satisfy r; and (2) How to compose multiple services to satisfy r when a matching service does not exist. We motivate our work through the following example.

Consider the four web services in Table 1.1, as illustrated in WSDL notation:

- Given the hotel, city, and state information, `findHotel` returns the address and zip code of the hotel.
- Given the zip code and food preference, `findRestaurant` returns the name, phone number, and address of the restaurant with matching food preference and closest to the zip code.
- Given the current location and food preference, `guideRestaurant` returns the address of the closest restaurant and its rating.
- Given the start and destination addresses, `findDirection` returns detailed step-by-step driving direction and a map image of the destination address.

Now, consider the following two requests from "State College, PA, USA":

- r_1: find the address of the hotel "Atherton", and
- r_2: find a *Thai* restaurant near the hotel "Atherton" along with a driving direction.

To fulfill r_1, invoking the web service `findHotel` is sufficient. That is, by invoking `findHotel`("Atherton", "State College", "PA"), one can get the address

```
<message name='findHotel_Request'>
  <part name='hotel' type='xs:string'>
  <part name='city' type='xs:string'>
  <part name='state' type='xs:string'>
</message>
<message name='findHotel_Response'>
  <part name='addr' type='xs:string'>
  <part name='zip' type='xs:string'>
</message>
```

(a) `findHotel`

```
<message name='findRestaurant_Request'>
  <part name='zip' type='xs:string'>
  <part name='foodPref' type='xs:string'>
</message> <message name='findRestaurant_Response'>
  <part name='name' type='xs:string'>
  <part name='phone' type='xs:string'>
  <part name='addr' type='xs:string'>
</message>
```

(b) `findRestaurant`

```
<message name='guideRestaurant_Request'>
  <part name='foodPref' type='xs:string'>
  <part name='currAddr' type='xs:string'>
</message> <message name='guideRestaurant_Response'>
  <part name='rating' type='xs:string'>
  <part name='destAddr' type='xs:string'>
</message>
```

(c) `guideRestaurant`

```
<message name='findDirection_Request'>
  <part name='fromAddr' type='xs:string'>
  <part name='toAddr' type='xs:string'>
</message>
<message name='findDirection_Response'>
  <part name='map' type='xs:string'>
  <part name='direction' type='xs:string'>
</message>
```

(d) `findDirection`

Table 1.1: Web service examples

of the hotel as "100 Atherton street" with the zip code of "16801." However, none of the four web services can satisfy r_2 alone. Both web services, `findRestaurant` and `guideRestaurant`, can find a Thai restaurant near the hotel, but cannot provide a driving direction. On the other hand, the web service `findDirection` can give a driving direction from one location to another, but cannot locate any restaurant. Therefore, one has to use a chain of web services to fully satisfy r_2. There are two possible methods to carry out this task. After obtaining the hotel address using `findHotel`, one can do either:

1. Invoke `guideRestaurant`("Thai", "100 Atherton street, 16801, PA") to get the address of the closest restaurant, e.g., "410 S. Allen St. 16802, PA." Then, invoke the web service `findDirection`("100 Atherton street, 16801, PA", "410 S. Allen St. 16802, PA") to get a driving direction.

2. Invoke `findRestaurant`("16801", "Thai") to get the address of the closest restaurant, e.g., "410 S. Allen St. 16802, PA." Then, invoke the web service `findDirection`("100 Atherton street, 16801, PA", "410 S. Allen St. 16802, PA") to get a driving direction.

1.3 Research Objectives

The main objectives in this research are: (1) Observation of existing web services; (2) Development of an effective web-service composition algorithm; and (3) Development of a web-service benchmark tool to test web-service composition and discovery algorithms.

1. The observation of real web services includes:

   - Observing the network properties of public web services and the ICEBE05 test sets in terms of the small-world network and scale-free network properties.
   - Exploiting the implications learned from above observation to develop a web-service benchmark tool with which we can generate web services to test web-service composition algorithms.

2. The development of an effective web-service composition algorithm includes:

   - The algorithm design, taking into consideration the fact that a large number of web services can be available, and diverse composition scenarios are possible depending on different service applications.
   - The performance tests in comparison with the other prominent AI planning algorithms over diverse test sets, including the EEE05 test set, the ICEBE05 test sets, and other sets with the proper capability to represent the diverse composition scenarios.

- Experimental validations based on the standard criteria for evaluating the performance of algorithms including effectiveness, efficiency, scalability, and robustness.
  - Effectiveness: it is defined as the ability to achieve stated goals or objectives, judged in terms of both output and impact. Our objective is to compose web services to generate the desired service with a minimum of web services.
  - Computational efficiency: we will measure how quickly our proposed algorithm generates the correct solutions in comparison with other AI planners as more data are applied to the problem.
  - Scalability: an algorithm has the scalability if it can adapt to increased workloads and continue to function well without exponential explosion of the running time. For this purpose, we will increase the size of test sets (the number of WSDL files) up to 50,000[2].
  - Robustness: an algorithm is considered robust if its performance remains relatively stable, with a minimum of variation, even though its application domain changes. We will use various test sets, including EEE05, ICEBE05 characterized by the tree structure, and our generated test sets featured by complex and random graph structure.

3. The development of an web-service benchmark tool which is expected to provide the following functions:
   - Flexible web services generation frameworks.
   - Supporting diverse web-service networks, including complex and random networks.

[2]The reason to choose the upper test size to be 50,000 is explainable. Four companies (IBM, Microsoft, NTT Com, and SAP) currently operate "Universal Business Repository"(UBR), a public instance UDDI. They manage around 50,000 replicated WSDL entries using their high performance machines. In fact, generating 50,000 WSDL files occupies as many as 3GB, no matter how we try to reduce the WSDL file size by maintaining only primal structure of WSDL. Considering these facts, we decide to conduct experiments by increasing the test size up to 50,000. Furthermore, we intend to show analytically that our proposal can run in polynomial running time in Section 6.

- Other functions to simplify the benchmark process, such as exporting test sets into AI planner readable files and generating test requests automatically.

1.4 Organization of the Thesis

In Chapter 2, the web-service composition problem is formally defined as three different models: (1) A STRIPS AI planning model, (2) A State Space model, and (3) A Network optimization problem. By means of these three formulations, we explain the intractability of the web-service composition problem. In Chapter 3, we extensively review the background literature on AI planning algorithms and web services matchmaking methods in the context of web-service composition. In particular, we suggest four facets to classify service composition problems and present a decision tree to help select the right solution to the various service composition problems. In addition, we compare existing AI planners with one simple motivating problem, and discuss their benefits and limitations. In Chapter 4, we analyze real web service sets such as public web services, the EEE05 test set, and the ICEBE05 test set with respect to diverse network properties (scale-free and small-world properties). Chapter 5 presents a novel web service composition benchmark tool titled WSBen. WSPR, an AI planning-based two step heuristic algorithm, is presented in Chapter 6. We show experimentally the benefits of the two step approach against the simple A* variant algorithms that use only a one-step forward search. Chapter 7 deals with testing and verification of our proposal, WSPR in comparison with other prominent AI planning algorithms. Chapter 8 suggests the application of semantic web-service composition in manufacturing, where we propose an ontology-based framework using semantic web-service technologies to secure the reliable and large scale interoperability among a design firm, manufacturers, and logistics providers. We also review existing semantic web-service technologies and propose a rough scenario that forms the motivating base for our proposed framework. Chapter 9 concludes the research and offers future research recommendations.

Appendix A discusses the WSBen's yTS framework which is not covered in Chapter 5. We discovered that the ICEBE05 test sets have interesting charac-

teristics with respect to their problem complexities. This discovery is presented in Appendix B. The web service monitoring and diagnosis is considered one of the promising research extensions by many researchers. Appendix C presents our previously proposed system titled MISQ, that aims at allowing users to analyze initial business processes and to obtain optimized parameters for implementing and monitoring their web-service composition. Appendix D and E provide WSPR and WSBen manual, respectively.

Figure 1.2 summarizes the road map of research and shows the organization of this thesis.

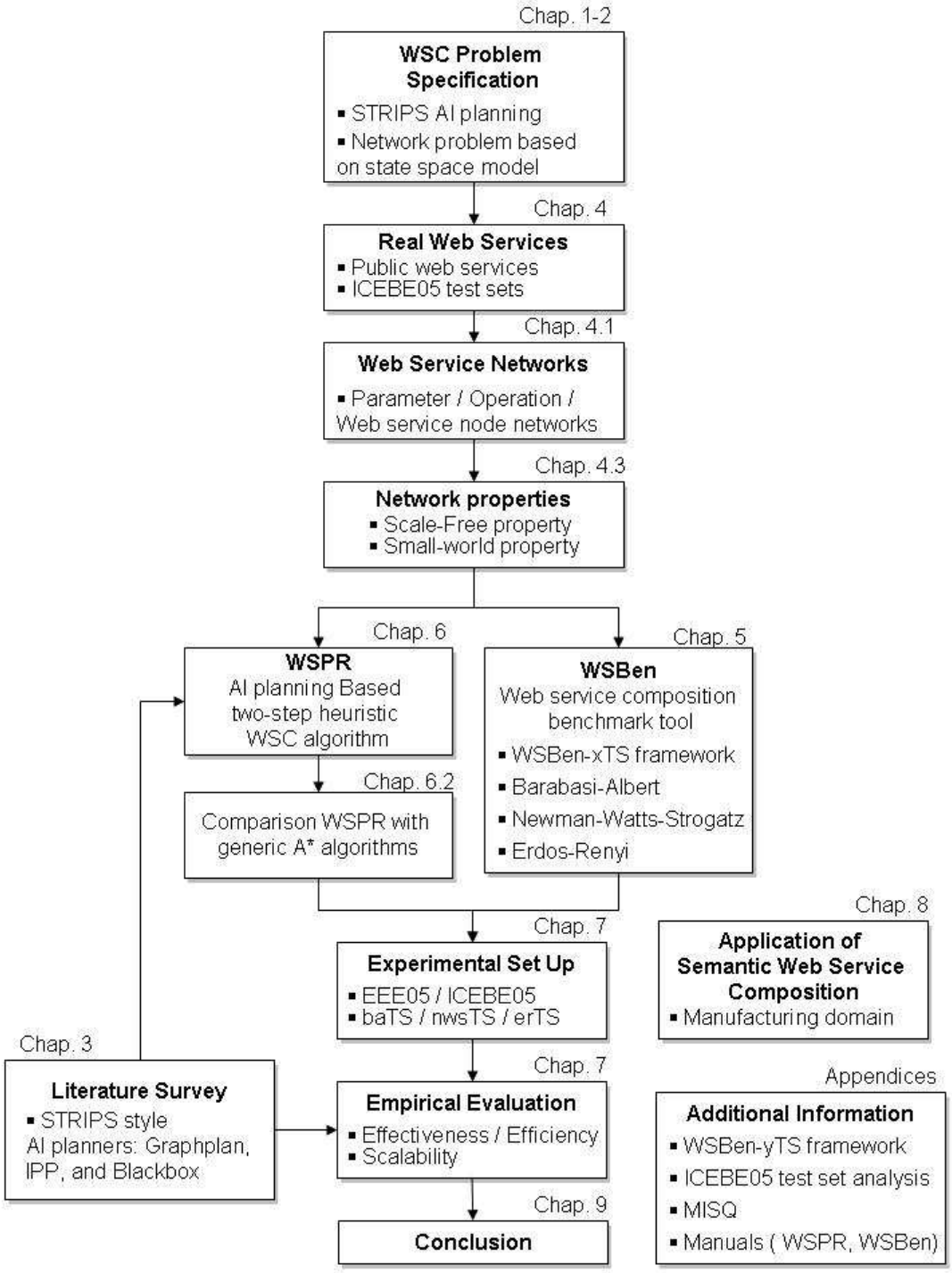


Figure 1.2: Thesis organization

Chapter 2

Problem Definition

A web service w has typically two sets of parameters: $w^i = \{I_1, I_2, \ldots\}$ for SOAP request (as input) and $w^o = \{O_1, O_2, \ldots\}$ for SOAP response (as output). When w is invoked with all input parameters, w^i, it returns the output parameters, w^o. We assume that in order to invoke w, all input parameters in w^i must be provided (i.e., w^i are mandatory).

Definition 2.0.1 (web-service Discovery Problem)**.** *Suppose that a request r has initial input parameters r^i and desired output parameters r^o. The web-service Discovery (WSD) problem is defined as to find a set of web services w, such that (1) $r^i \supseteq w^i$ and (2) $r^o \subseteq w^o$.*

With a simple look-up table, the WSD problem can be easily solved. Therefore, in the remainder of the thesis, we focus on the case where no single web service can fully satisfy the request r, and therefore one has to compose multiple web services. This type of problem is referred to as the *Web-Service Composition (WSC)* problem, and can be formally defined using a planning problem in the STRIPS[1] model of the 4-tuple: $\Pi = \langle P, W, r^i, r^o \rangle$, where:

(1) P is a set of parameters. In the motivating example in Chapter 1, P = {"hotel-name", "hotel-city","hotel-address", ...},

(2) W is a set of web services. In the motivating example, W={"findHotel", "findRestaurant",...},

[1]STRIPS [79] is the first major AI planning system which represents actions in terms of their preconditions and effects. For details, see Chapter 3

(3) $r^i \subseteq P$ is the initial input parameters, and

(4) $r^o \subseteq P$ is the desired output parameters.

Note that $\Pi = \langle P, W, r^i, r^o \rangle$ is a propositional STRIPS planning in which an initial state is a finite set of ground atomic formulas, indicating that the corresponding conditions are initially true and that all other relevant conditions are initially false. In addition, the pre-conditions and post-conditions of an operator, as well as the goals are the ground literals [24]. Figure 2.1 illustrates the STRIPS model of the motivating example.

We can transform a STRIPS model Π into a state space model $\Psi = \langle S, s_0, S_G, \Omega(.), f, c \rangle$, where:

(1) The state, $s \in S$ is a collection of parameters in P,

(2) The initial state $s_0 \in S$ is such that $s_0 = r^i$,

(3) The goal states $s \in S_G$ are such that $r^o \subseteq s$,

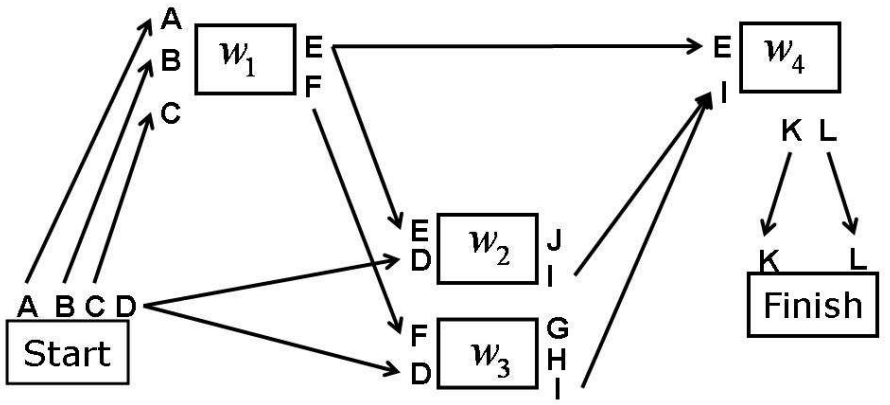
(4) $\Omega(s)$ is a set of web services $w \in W$ such that $w^i \subseteq s$. That is, w can be invoked or applicable in the state s,

(5) The transition function $f(w, s) = s'$ that maps a state s into a state s' such that $s' = s \cup w^o$ for $w \in \Omega(s)$, and

(6) $c(w)$ is the invocation cost of w.

A solution of the state model is a finite sequence of web services $w_1, w_2, \ldots, w_n$, where a sequence of states $s_0, s_1, \ldots, s_n$ exists, such that $s_i = f(w_i, s_{i-1})$ for $i = 1, \ldots, n$, $w_i \in \Omega(s_{i-1})$, and $s_n \in S_G$. Based on Ψ, the WSC problem can be formally defined as follows:

Definition 2.0.2 (web-service Composition Problem)**.** *Suppose that a request r has initial input parameters r^i and desired output parameters r^o. The web-service Composition (WSC) problem is to find a finite sequence of web services, $w_1, w_2, \ldots, w_n$ such that (1) w_i is invoked sequentially from 1 to n; (2)$(r^i \cup w_1^o \cup \ldots \cup w_n^o) \supseteq r^o$; and (3) the total cost $\sum_{i=1}^{n} c(w_i)$ is minimized.*



W: **web services set**

findHotel	w_1
findRestaurant	w_2
guideRestaurant	w_3
findDirection	w_4

P: **parameter set**

hotelName	A	restaurantName	G
hotelCity	B	restaurantPhonenumber	H
hotelState	C	restuarantAddress	I
foodPreference	D	restaurantRate	J
hotelAddress	E	mapHotelRestaurant	K
hotelZipcode	F	directionHotelRestaurant	L

Figure 2.1: STRIPS model of the motivating example.

Except for attempting to minimize the total cost, the WSC problem is a kind of classical planning problem which refers generically to planning for restricted state-transition systems [44]. An issue of interest in the AI planning community primarily concerns scenarios which allow interleaving of actions from different subplans within a single sequence. By contrast, the WSC problem can be considered to be the information gathering problem [62], where web services represent information sources and interleaving between web services is not found. This enables a highly specialized planning algorithm. The only preconditions to web services are knowledge preconditions. Furthermore, there are no sibling sub-goal interactions, such as those characterizing the Sussman anomaly[2] [79]. For that reason, a WSC algorithm models the world state as an information state, which is a description of the information collected by the algorithm at a particular stage in composition [94]. Therefore, the WSC problem is a relatively simple problem from a planning perspective, but has quite different characteristics than a usual planning problem. Classical planning problems (e.g., blocks world problem) have generally

[2]No combination of plans for the two individual goals can solve the conjunction of the two goals.

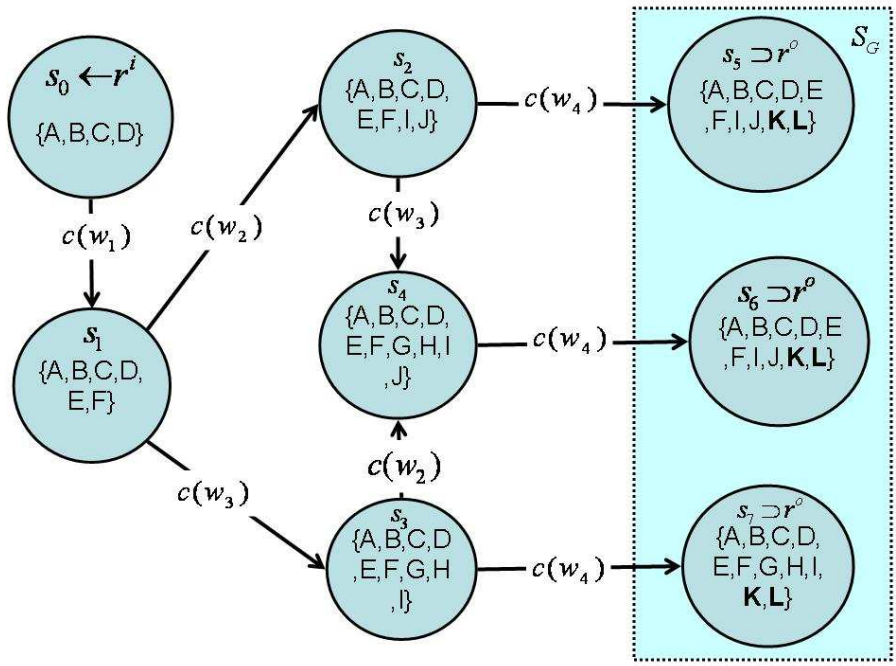


Figure 2.2: $G_s(V_s, E_s)$ of the motivating example.

considered a small number of actions (e.g., move block) under the assumption of a large number of objects (e.g., hundreds of blocks). On the contrary, WSC problems generally deal with a large number of actions (e.g., hundreds of travel agent services) with a limited number of objects involved (e.g., registering one hotel).

Definition 2.0.3 (State Node Network)**.** *A state node network is a directed graph* $G_s(V_s, E_s)$*, where* $s_i (\in V_s)$ *represents a state in* S*, and* E_p *is a set of directed edges*[3] (s_i, s_j) *that connects ordered pairs of* $s_i \in V_s$ *and* $s_j \in V_s$ *. Every arc* $(s_i, s_j) \in E_s$ *is weighted by the invocation cost* $c(w)$*, where* $w \in \Omega(s_i)$ *and* $s_j = f(w, s_i)$ *(i.e.,* $s_j = s_i \cup w^o$*).*

Given a network $G_s(V_s, E_s)$, we consider paths in which arcs are traversed only in the forward direction. The cost of a path is the sum of the costs of the associated arcs. Our interest is to obtain the shortest path between s_0 and $s_n (\in S_G)$. Figure 2.2 illustrates the $G_s(V_s, E_s)$ of the motivating example. Note that s_5, s_6, and s_7 contain the goal parameters, "K" and "L", where "K" and "L" imply "mapHotelRestaurant" and "directionHotelRestaurant", respectively.

[3]We will use the terms *directed edge* and *arc* in the same meaning

Therefore, s_5, s_6, and s_7 belong to S_G. It is evident that if $c(w_i) = 1$[4] for all $w_i \in W$, then the shortest paths are: (1) $s_0, s_1, s_2, s_5 (w_1 \rightarrow w_2 \rightarrow w_4)$; and (2) s_0, s_1, s_3, s_7 $(w_1 \rightarrow w_3 \rightarrow w_4)$. Indeed, the problem of finding the shortest paths from r^i to r^o in G_s can be formulated as the minimum cost flow problem.

Let b_i denote the amount of flow that enters the network at node $s_i \in V_s$, $i = 0, \ldots, n$. Let $fl_{i,j}$ denote the amount of flow which is greater than 0[5] on arc $(s_i, s_j) \in E_s$. If $b_i > 0$, the node is a source that supplies b_i units of flow. If $b_i < 0$, the node is a sink that demands b_i units of flow. Suppose $s_0 = r^i$, and $s_n \in S_G$. Although we are interested in obtaining the shortest path between s_0 and s_n, it is also possible to consider the issue of finding the shortest path from all states to a given goal state by setting $b_i = 1$, for $i = 0, \ldots, n-1$ and $b_n = -n$. Simply, this will put a supply of one unit at every other non-goal state node and a demand of n at the goal state node so that the total supply and demand are equal. Suppose that a cost of $c_{i,j}$ per unit flow on arc $(s_i, s_j) \in E_s$ is $c(w)$, where $s_j = f(w, s_i)$. Then, the minimum cost flow problem is as follows:

$$minimize \sum_{(s_i,s_j)\in E_s} c_{i,j} fl_{i,j} \tag{2.1}$$

subject to

$$\sum_{(s_i,s_j)\in E_s} fl_{i,j} - \sum_{(s_k,s_i)\in E_s} fl_{k,i} = b_i \qquad \forall s_i \in V_s, i = 0, \ldots, n \tag{2.2}$$

$$fl_{i,j} \geq 0 \qquad \text{for } \forall (s_i, s_j) \in E_s \tag{2.3}$$

[4]Pricing web services has been a critical issue. The emergence of web services has required an ideal pricing mechanism. A widely accepted method is dynamic pricing, which is a general equilibrium model with a resource-price structure. In this research, we assume that all web services have an identical equilibrium market price of 1, meaning that at price 1, the quantity demanded equals quantity supplied. This assumption is the same as in the previous web-service composition contest such as EEE05 and ICEBE05. In a real system, however, determining price of a web service is subject to the Quality-of-Service (QoS) requirements of service requests. A better pricing model could be constructed based on real-market surveys.

[5]$fl_{i,j}$ will automatically be integer due to the unimodularity property of network flow programming. If right-hand side values in all constraints are integers (as in our case where b_i=0 or 1), and if all of the pivot operations are simple additions and subtractions, then we can guarantee that the solution values of the variables at the optimum will also all be integers. This is known as the unimodularity property.

This expression indicates that flows must be feasible and each node conserves flow. Specifically, Objective function (2.1) indicates that we are interested in getting the shortest plan. Constraint set (2.2) called as the "flow conservation equation" indicates that the flow may be neither created nor destroyed in the network. In the conservation equation, $\sum_{(s_i,s_j)\in E_s} fl_{i,j}$ represents the total flow out of node s_i, while $\sum_{(s_k,s_i)\in E_s} fl_{k,i}$ indicates the total flow into node s_i. This equation requires that the net flow out of node s_i, $\sum_{(s_i,s_j)\in E_s} fl_{i,j} - \sum_{(s_k,s_i)\in E_s} fl_{k,i}$, should be equal to b_i. Note that if $b_i < 0$, then there should be more flow into i than out of i. Feasible flows exist when $\sum_{i=0}^{n} b_i = 0$.

This minimum cost flow problem can have a dual problem. With $b_0 = b_1 = \cdots = b_{n-1} = 1$, the dual problem with $\lambda_n = 0$ is

$$maximize \sum_{i=0}^{n-1} \lambda_i \tag{2.4}$$

subject to

$$\lambda_i \leq c_{i,j} + \lambda_j \qquad \text{for } \forall (s_i, s_j) \in E_s \tag{2.5}$$

λ_i refers to the cost associated with the path from s_i to s_n. This maximization problem suggests that in the optimal solution, λ^*, if all components are fixed except for λ_i^*, then λ_i^* tends to become as large as possible and is subject to the feasibility Constraint 2.5. Therefore, λ_i^* satisfies the following Bellman equation with $\lambda_n^* = 0$

$$\lambda_i^* = \min_{(s_i,s_j)\in E_s} \{c_{i,j} + \lambda_j^*\}, \qquad i = 0, \ldots, n-1 \tag{2.6}$$

λ_i^*s are also called *labels*, and there are polynomial algorithms to solve the equation outlined in 2.6, such as the label setting algorithm (when $c_{i,j}$ is non-negative) or label-correcting algorithm [30]. However, problems exist with this idea. These algorithms are polynomial in the size of nodes $|V_s|$, but the number

of nodes are exponential in the number of parameters $|P|$. This is because nodes are states, and the state is a collection of parameters in P. Before we proceed to investigate the complexity of the WSC problem in detail, we need to introduce two matching operations described below.

Definition 2.0.4 (Full matching)**.** *Suppose that a state $s \in S$ is given. Let a web service $w_1 \in \Omega(s)$. If for $w_2 \in W$, $w_1^o \supseteq w_2^i$, then w_1 can "fully" match w_2.*

Definition 2.0.5 (Partial matching)**.** *Suppose that a state $s \in S$ is given. Let a web service $w_1 \in \Omega(s)$. If for $w_2 \in W$, $(w_1^o \not\supseteq w_2^i)$ and $(w_1^o \cap w_2^i \neq \emptyset)$, then w_1 can "partially" match w_2.*

In the motivating example, `findHotel` partially matches `findRestaurant` and `guideRestaurant`. In turn, `findRestaurant` and `guideRestaurant` partially match `findDirection`. However, if both `findHotel` and `findRestaurant` are composed, then `findDirection` can be fully matched. When only full matching is considered in the WSC problem, it can be seen as a single-source shortest path problem which is defined over $G_s(V_s, E_s)$ because the total number of input and output parameter sets is $2 \times |W|$ thereby $|V_s| = 2 \times |W|$; each parameter set can be considered as one atomic parameter. Thus, the computational complexity becomes tractable. On the other hand, when both full and partial matching must be considered concurrently, the problem becomes a decision problem to determine the existence of a solution of k operators or less for propositional STRIPS planning [79] with restrictions on negation in pre- and post-conditions. This type of STRIPS planning problem is proved to be NP-complete by Bylander [24]. We can revise the result in Bylander [24] to prove that the WSC problem, including full and partial matching operations, is NP-complete as follows:

Theorem 2.0.6. *The web-service composition problem with full and partial matching operations is NP-complete.*

Proof. In this proof, the WSC problem is considered to be a decision problem of determining whether an instance of propositional STRIPS planning has a solution of k or fewer operators, where k is given as part of input. The first part of this proof focuses on proving that the WSC problem is NP, and the second part concentrates

on proving that the WSC problem is NP-hard by building a polynomial time reduction procedure from 3SAT[6] [94] to the WSC problem.

NP: Web services without negative effects can never negate a condition. Thus, a previous state is always a subset of succeeding states. Also, web services within a service sequence that have no effect can always be removed. Hence, if a solution exists, the length of the smallest solution can be no longer than the total number of web services. Thus, the WSC problem is in NP because only a linear number of nondeterministic choices are required. In other words, a solution to a WSC problem can be verified in polynomial time.

NP-hard: 3SAT can be reduced to the WSC problem in polynomial time. It is also written in 3SAT $\leq_p$ WSC. Let $U = \{u_1, u_2, \ldots, u_m\}$ be the set of variables used in an instance of 3SAT. Let n be the number of clauses. The equivalent WSC problem to the instance of 3SAT can be constructed using the following types of parameters or conditions.

- $T(k)$: if u_k=true is selected, $T(k)$ is true; otherwise false.
- $F(k)$: if u_k=false is selected, $F(k)$ is true; otherwise false.
- $V(k)$: if some value (i.e., either true or false) for u_k has been selected, then $V(k)$ is true; otherwise false.
- $C(j)$: if the jth clause is satisfied, it is true; otherwise false.

The initial state and goal can be specified as:

- $r^i = \emptyset$
- $r^o = V(1) \wedge V(2) \wedge \cdots \wedge V(m) \wedge C(1) \wedge C(2) \wedge \cdots \wedge C(n)$

That is, the goal is to select a value for each variable (i.e., V) and satisfy each of the clauses. For each variable u_k, four web services are needed:

- wt_k: $wt_k^i = \emptyset$ and $wt_k^o = T(k)$.

[6]Deciding whether a given boolean formula in conjunctive normal form has an assignment that makes the formula "true". A statement is in 3 conjunctive normal form if it is a conjunction (sequence of ANDs) consisting of one or more conjuncts (clauses), each of which is a disjunction (OR) of three or less literals. In 1971, Cook showed that the problem is NP-complete.

- wf_k: $wf_k^i = \emptyset$ and $wf_k^o = F(k)$.
- wv_{k1}: $wv_{k1}^i = T(k)$ and $wv_{k1}^o = V(k)$.
- wv_{k2}: $wv_{k2}^i = F(k)$ and $wv_{k2}^o = V(k)$.

These four web services are used to select a value for u_k, and to ensure that a value has been assigned to u_k. Note that nothing prevents the double selection of both true and false for one variable. To prevent the double assignment, we require two more web services, wc_{j1} and wc_{j2}. If the jth clause contains a variable u_k, wc_{j1} is needed. On the contrary, if the jth clause contains a negated variable $\bar{u_k}$, wc_{j2} is needed:

- wc_{j1}: $wc_{j1}^i = T(k)$ and $wc_{j1}^o = C(j)$ where u_k is contained in the jth clause.
- wc_{j2}: $wc_{j2}^i = F(k)$ and $wc_{j2}^o = C(j)$ where $\bar{u_k}$ is contained in the jth clause.

If the 3SAT problem is satisfiable, then r^o is true because all $V(k)$ and $C(j)$ must be true. On the other hand, if the 3SAT problem is not satisfiable, then at least, one of $C(j)$ must be false, resulting that r^o is false.

In addition, if the 3SAT formula is satisfiable, then $2m + n$ web services are sufficient, because:

- Only one value between true and false, is selected for each variable. Therefore, m web services are required.
- m number of web services are required to ensure that m number of variables are set to be true or false value.
- n number of web services are required to indicate that n number of clauses are determined to either be true or false.

On the contrary, if the 3SAT formula is not satisfiable, then both values must be selected for some variables to achieve the goals, implying that more than $2m + n$ web services are needed. Thus, the 3SAT formula is satisfiable if and only if there is a web-service composition of size $k = 2m + n$ □

Therefore, when the number of web services to search is not small, finding an optimal solution to the WSC problem (a sequence of web services from r^i to r^o) is prohibitively expensive, leading to the use of approximate algorithms instead. To address this intractable WSC problem, we will suggest a polynomial-time AI-planning based heuristic algorithm in Chapter 6.

Chapter 3

Background Literature Survey

In this chapter we present an overview of previous research concerning the current research topic. First, we suggest four facets to classify WSC problems, and present a decision tree to help select the right solution to various WSC problems. Second, we compare existing AI planners with one simple motivating WSC problem, and discuss their benefits and limitations. In addition, we study existing web services matchmaking methods including the exact matchmaking and semantic matchmaking.

3.1 Classification of WSC Problems and Related Research Work

We can classify the WSC problem using the following four facets :

- **Manual vs. Automatic Workflow Composition**: In building workflows by means of web services, one can either do (1) Manual composition in cooperation with domain experts; or (2) Automatic composition by software programs. In the manual approach, human users who know the domain well (e.g., domain ontology) select the proper web services and weave them into a cohesive workflow. Although users may rely on GUI-based software to facilitate the composition, in essence, it is a manual and labor-intensive task, and thus is not appropriate for large-scale WSC.

- **Deterministic vs. Stochastic Environment**: If we view the WSC as a design-oriented process by ignoring the stochastic real-world environment, AI-inspired planning techniques are suitable for the WSC. However since the classical STRIPS-style planning algorithms assume deterministic behavior of web services, additional overhead is required to monitor execution time to recover from unexpected behavior of web services, such as failure. There are some research efforts to suggest approaches for dynamically composing web services in run-time with considerations to the stochastic environment.

- **Simple vs. Complex Operator**: The simple WSC involves only two types of matching operations as described in Chapter 2: (1) Full-matching operator; and (2) Partial-matching operator. If WSC involves the two operators alone, it can be considered to be a linear planning, as there are no sibling-sub goal interactions such as those characterizing the Sussman anomaly. More complex WSC, however, can use other operators (e.g., OR, XOR, NOT) or constraints (e.g., request prefers sources in Asia to the ones in Europe) in both sequential and parallel modes. Note that throughout this thesis, WSC means the simple WSC involving only the fully- and partial matching operations.

- **Small vs. Large Scale**: The general WSC problem to find an optimal workflow can be formulated as an AI planning problem, into which the satisfiability (SAT) problem can be reduced [107]. We know that the SAT problem is NP-complete and therefore is unlikely to have a polynomial algorithm for the WSC problem. Note that we defined the WSC problem as an optimization problem in Chapter 2. However, it is true that many solutions are currently available with the capability of dealing with large scale problems. This is possible because of their polynomial-time nature to approximate optimal solutions.

In Figure 3.1, we present a taxonomy to help choose the right solution using the aforementioned four facets. The manual composition approach can rely on software programs that have functions to bind manually-generated workflows to the corresponding concrete resources. To that end, METEOR-S [104] and Proteus [45] were suggested. Kepler [5], in particular, provides a scientific workflow editor,

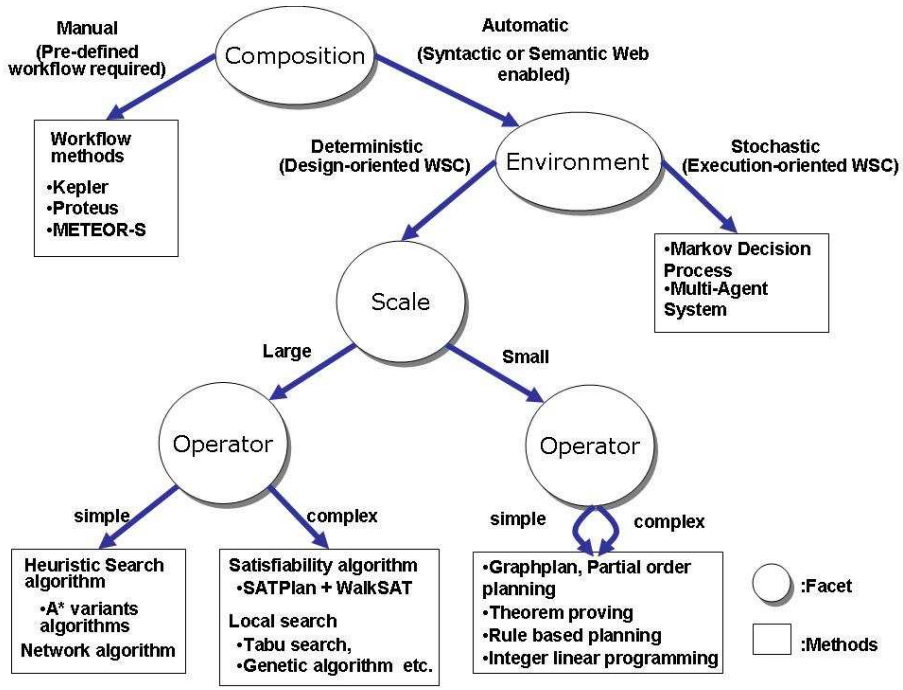


Figure 3.1: Taxonomy for classifying WSC problems and solutions

which allows scientists to effectively query and compose distributed data sources on the Grid. Thus, it is possible to build scientific workflows across diverse scientific domains for analysis and modeling tasks. METEOR-S, Proteus, and Kepler adopt the idea of semi-automatic service composition and indicate the trend that GUI-based software and human experts can work together to generate composite services. However, we believe that by leveraging the various planning-based solutions of AI community, one can solve the WSC problem better. If one knows that an automatic composition is plausible for the given WSC problem, then she or he may apply planning-based automatic composition solutions first to get the initial set of candidate workflows, and subsequently perform additional fine-tuning.

Regarding AI planning solutions for WSC in Figure 3.1, STRIPS [79] is the first major AI planning system which represents actions in terms of their preconditions and effects. Additionally, it describes the initial and goal states as conjunctions of positive literals. Graphplan [19] is a general purpose planner for STRIPS-style domains, and it is based on ideas used in graph algorithms. Given a problem statement, Graphplan uses a backward search to extract a plan (so-called, "plan

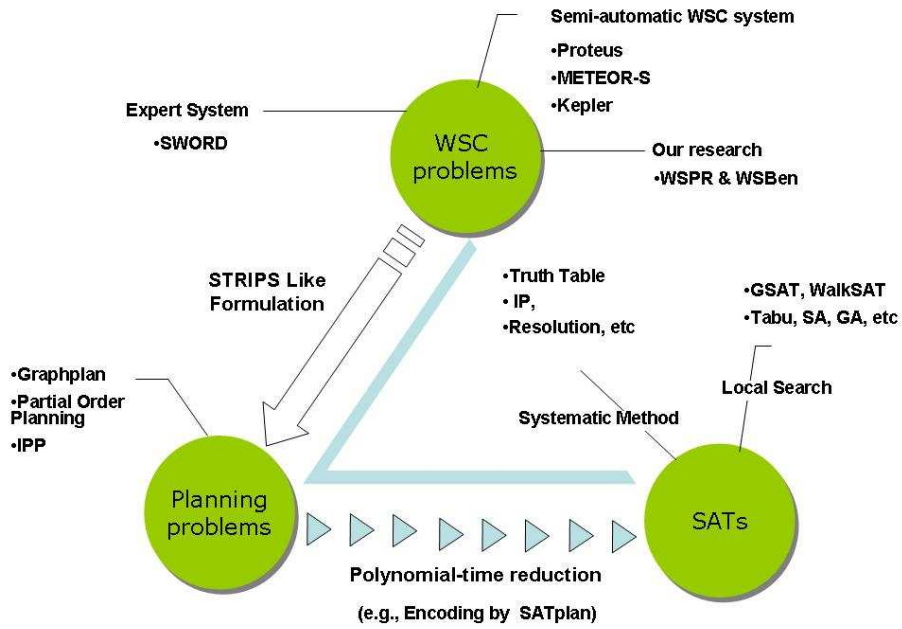


Figure 3.2: Relationships between WSC problems and other related problems with their solutions

graph") and allows for some partial ordering among actions. As a satisfiability approach for the planning problems, the SATPlan algorithm [55] was introduced. SATPlan belongs to the greedy local search methods for solving SAT problems. The SATPlan algorithm translates a planning problem into propositional axioms and applies a satisfiability algorithm to find an assignment that corresponds to a valid plan. An excellent survey of modern planning algorithms and their application to WSC problems is available [81, 92, 121]. Furthermore, we organize the problems related to the WSC problem along with possible solutions as shown in Figure 3.2.

However, it must be noted that these AI planning-based algorithms cannot handle the case when a semantic match is desired. In other words, for the purpose of the semantic match, a promise to use agreed-upon ontology is required between participating parties in advance. From a planning objective point of view, Graphplan and SAT-based planning systems have the same objective of minimizing the number of time steps, but not necessarily the number of actions, to reach a goal. However, through an integer linear programming (ILP) formulation, other various

QoS factors (e.g., response time, service cost, or availability of sources) can be incorporated and optimized. Clearly, an exhaustive search is unrealistic because combinatorial explosion renders it impractical. When the scale is large but the WSC problem is free of negation, we can use heuristic search algorithms like A* variant algorithms [84]. This will be discussed in Chapter 6 in detail. SWORD [88] is a rule-based expert system which can automatically determine whether a desired composite service can be realized using existing rules. However, considering the fast growth of web services, building a full knowledge base by converting all web services into axioms will be greatly expensive.

Different from the deterministic environment composition above, there are some studies to address the non-determinism inherent in real-world web services. Markov decision processes can be used to utilize a stochastic optimization framework in this context, but the main focus is on the abstract-level strategies instead of the implementation-level details [36]. A multi-agent perspective can be used to compose adaptive workflows in a dynamic environment [23]. However, it is likely to be expensive to enable web services with agent-level intelligence. In the database community, one of the recent works to support an automated service discovery was attempted by using Information Retrieval (IR)-like similarity search [35]. The emergent need of workflows to model e-service applications makes it essential that workflow tasks be associated with web services. As a result, research efforts have been carried out to enhance workflow systems in their support of web service composition [99].

Regarding the service matching and composability, Li and Horrocks [65] use DAML-S to design a service matchmaker. DAML-S is used to represent knowledge for promoting service capability matching. A description logic (DL) based on DAML-S is used to implement matchmaking details. Medjahed et al. [72] propose an ontology-based framework for the automatic composition of web services. A technique to generate composite services is based on high-level declarative descriptions and applied for implementing an E-government application offering customized services to indigent citizens. Sirin et al. [103] demonstrate how an OWL reasoner can be integrated with an AI planner to overcome automatic web-service composition problems. They identify the challenges of writing the service descriptions and reasoning about them when an expressive language as OWL is used.

As an independent research branch, web-service quality testing has been established where quality and trustworthiness of web services are considered [130]. It is very unlikely that a business organization will dynamically select a partner from the Internet merely based upon the information found from some public registry without having a high confidence regarding it. Therefore, the test of quality and trustworthiness of web-services based software is a critical work for the success of web-service paradigms.

There is an emerging consensus that the ultimate challenge is to make web services automatically tradable and usable by artificial agents in their rational, pro-active interoperation in the next generation of the web [40]. This may be solved by creating effective frameworks, standards and software for automatic web-service discovery, execution, composition, inter-operation and monitoring [76]. In industries, only initial and partial solutions of the ultimate problem are provided.

Existing de-facto standards for web service description (WSDL) [114], publication, registration and discovery (UDDI) [113], binding, invocation, and communication (SOAP) [112] provide merely syntactic capabilities and do not completely solve the ultimate composition problem. More recent research and standardization activities of the DARPA DAML community resulted in offering the semantic service markup language DAML-S [8] based on RDF platform.

3.2 Overview of Matching Approaches

The WSC problem needs to integrate information from heterogeneous sources. Since individual web services are created in isolation, their vocabularies have often problems with abbreviations, different formats, or typographical errors. Furthermore, two terms with different spellings may have the same semantic meaning, and thus are inter-changeable (e.g., "price" and "fee"). Conversely, two terms with the same spelling may have different meanings (e.g., "title" may mean either "book title" or "job title").

In response to these challenges, researchers have developed diverse matching schemes. Consider that x and y are data objects (e.g., web service parameters; individual record field) with a vector of attributes: $x = (x_1, x_2, \ldots, x_k)$, and $y = (y_1, y_2, \ldots, y_k)$, where k is the number of attributes. We can quantify the

"similarity" between x and y by a distance function, $d(x, y)$ with properties:

(1) $d(x, y) \geq 0$, where equality holds if and only if $x = y$,

(2) $d(x, y) = d(y, x)$, the symmetric property,

(3) $d(x, y) \leq d(x, z) + d(z, y)$, the triangle inequality.

By using different distance functions $d(x, y)$, one can employ different matching approaches. In general, matching approaches may fall into three categories:

(1) Approach-1: exact match using syntactic equivalence,

(2) Approach-2: approximate match using distance functions, and

(3) Approach-3: semantic match using ontologies (e.g., RDF [33] and OWL [109]).

In Approach-1, two objects x and y are deemed to be a match if and only if $x = y$. However, with this approach, two objects with slightly different representations (e.g., "William Jefferson Clinton" and "Bill Clinton") cannot be matched. For this reason, in Approach-2, if two objects are similar enough according to some distance function (i.e., $d(x, y)$ is above some threshold), they are deemed to be a match. Bilenko et al. [16] reported different pros and cons of popular distance functions, including TF-IDF, Jaccard, etc. For details of distance functions, refer to [28].

Although Approach-2 is much more flexible than Approach-1, it is not sufficient enough to identify that "price" and "fee" are interchangeable. In response, researchers have created the vision of the semantic web, where data has structure and ontologies describes the semantics of the data. Based on the semantic web foundation, Approach-3 can address the ontology-matching problem to find semantic mapping between two ontologies, specified by languages like DAML+OIL [108], OWL, and RDF. Note that in this thesis, the choice of approach of matching is irrelevant to the WSC problem. We assume that these matching tasks are pre-processed and pre-selected.

When it comes to the web service matchmaking, current solutions are based on the keyword matching supported by the category-browsing of UDDI. However, the keyword-based matching considers only the name of web services and ignores their

real functions. To remedy this limitation, researchers have developed a set of methods which assess the similarity of web services to achieve matchmaking. Wu [127] suggested a matchmaking process based on a lightweight semantic comparison of signature specifications in WSDL by means of several assessment methods. Wang and Stroulia [116] assessed the similarity of the requirement description of the desired service with the available services via the semantic information-retrieval (IR) method and a structure-matching approach. Maedche and Staab [67] provide multiple-phase cross-evaluation to assess the similarity between two different ontologies. Gilles [46] introduces several similarity measures in order to extract a new complex concept into an existing ontology by similarity rather than by logic subsumption. Cardoso and Sheth [26] introduces a similarity function to determine similar entity classes by using a matching process over synonym sets, semantic neighborhoods, and distinguishing features that are classified into parts, functions, and attributes.

Besides the web service matchmaking field, similarity measures have been widely used in information systems [47, 63, 106], cognitive science, databases [18, 22], software engineering [64] and AI field [53]. An excellent survey of modern matchmaking algorithms and their applications to the web service matching field is available in [127]. Shvaiko and Euzenat [101] introduced a new classification method by enhancing previous survey works [91]. Their work provides useful texonomy and criteria for classifying many different recent schema and ontology matching techniques, pointing which part of the solution space the techniques can cover.

3.3 Comparative Illustration

In this section, we illustrate three selected automatic-composition algorithms for the WSC problem and discuss their benefits and limitations. Among many state-of-the-art proposals, we chose the following three for their impact on other solutions: Graphplan, SATPlan, and Integer Linear Programming (ILP).

Traditionally, popular algorithms for STRIPS planning problems include total-order planners, like Prodigy [90], and partial-order planners, like UCPOP [12]. In another approach, Graphplan was introduced and significantly outperformed

previous approaches [19]. Many researchers argue that SAT solvers are more robust and allow for more expressiveness than specialized planning algorithms. In fact, a variety of SAT algorithms are available to obtain results competitive with those of Graphplan. SATPlan was suggested to convert STRIPS planning problems directly into a conjunctive normal form (CNF), to which those SAT solvers were applied. However, SATPlan sometimes requires hand-coded axioms to fine-tune its performance. SATPlan can be classified into a satisfiability-based approach to solving STRIPS planning problems.

The attempt to combine the strength of Graphplan in defining a search space and the strength of modern research into SAT solvers resulted in Blackbox. Blackbox first uses a planning graph similar to Graphplan, and then converts the planning graph into a CNF formula to be fed into a SAT solver, Walksat [97]. Since Blackbox obtains the CNF formula by converting a planning graph instead of the STRIPS planning problems directly, Blackbox shows improvement over both Graphplan and SATPlan, in addition to the benefits of automatic SAT encoding without needing hand-coded axioms. Consequently, Graphplan and SATPlan are the origin of Blackbox. We will compare Blackbox and other AI planners in Chapter 7. Regarding the Walksat, this algorithm belongs to the incomplete local search algorithms. The algorithm seeks for an assignment of the variables that satisfies a given formula. The task is done by means of two strategies: a random walk strategy and a greedy strategy. The algorithm starts with a random assignment of the variables. Then it selects an unsatisfied clause at random and changes (flips) the value of one of the variables to satisfy the clause. It will do this flipping process until no clause is unsatisfied or until a maximum number of tries is reached. Walksat adds a random noise; with certain probability p, the algorithm flips a variable in the clause randomly (the random-walk strategy), and with probability $1 - p$, the algorithm flips the value of the variable whose change will result in the least number of unsatisfied clauses (the greedy strategy)

ILP which has a rich history in the operational research community, has shown good performance for AI planning problems [107]. Moreover, ILP naturally allows the incorporation of various constraints and objectives into the planning domain.

Figure 3.3 represents a simplified version of the motivating example of Chapter 1 in terms of STRIPS-style representation, where $\Pi = \langle P, W, r^i, r^o \rangle$ is: (1)

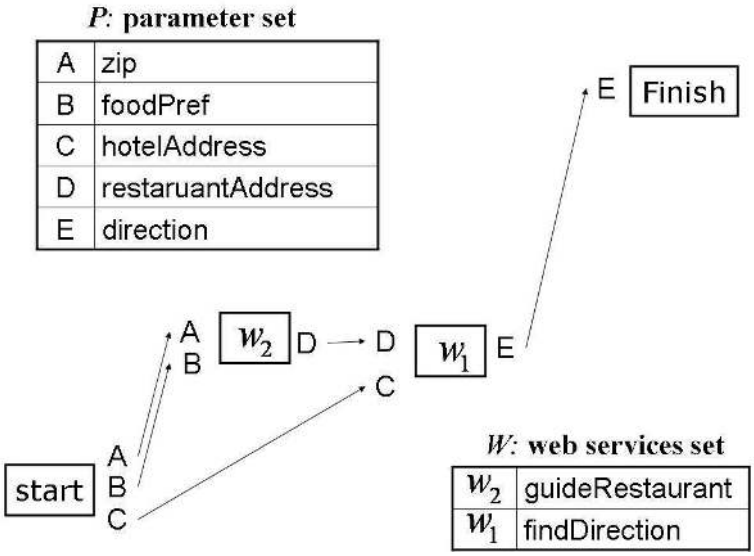


Figure 3.3: STRIPS representation.

P={A,B,C,D,E}; (2) W={w_1,w_2}; (3) r^i={A,B,C} ; (4) r^i={E}. We can use the STRIPS-style notation for describing the transitions. For instance, `findRestaurant` action has the precondition, "zip" and "foodPref" and the effect. In the following, we will illustrate how these three methods attempt to solve the example in Figure 3.3 differently.

3.3.1 Graphplan-based planning

The operation of Graphplan consists of two phases. First, a forward search is used to build a plan graph and estimate the costs of all parameters from the initial state r^i. Second, a regression search is performed using those measures for guidance. These two phases are in correspondence with the two step approach of our proposed WSC algorithm, WSPR[1], where we estimate the costs of parameters in a first phase, and exploit those costs for the backward search in the second phase. However, in the second phase, Graphplan and WSPR are quite different, as each uses a different type of heuristic.

Graphplan can be understood as a heuristic search planner with an admissible heuristic function and search algorithm. The search algorithm of Graphplan can

[1]WSPR is pronounced as "whisper". WSPR is introduced in Chapter 6

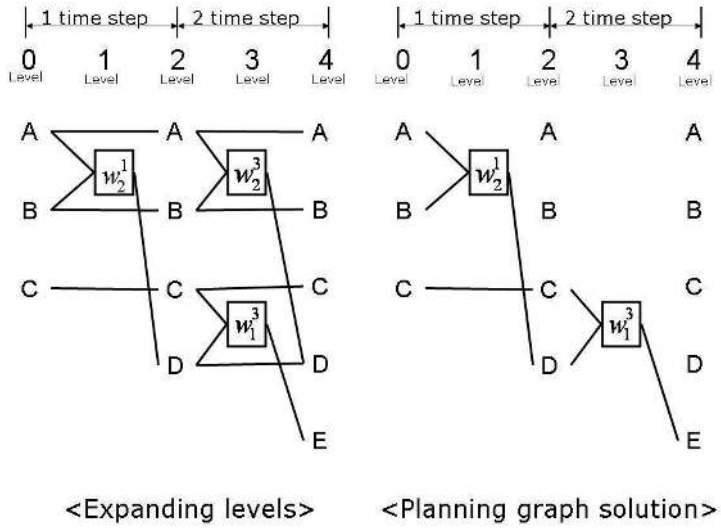


Figure 3.4: Planning using Graphplan.

be considered to be a version of Iterative Deepening A* (IDA*) [59] because of the admissible heuristic function. The form of IDA* will be discussed in Chapter 6. Moreover, the IDA* search algorithm of Graphplan is more efficient than general IDA* because of the planning graph. Specifically, Graphplan searches a parallel regression space. It has been reported that Graphplan is an order-of-magnitude faster than other planners on parallel domains [50]. However, the plan graph restricts Graphplan to IDA* search, and it cannot be easily adapted to best-first searches. Due to this restriction, Graphplan is unable to produce suboptimal solutions quickly. However, it is trivial in general heuristic algorithms where simple modification of their search algorithms achieves the desired effect. In addition, in serial problems in which sub-goals are mostly independent, like Gripper[2] [70], Graphplan yields poor results because the heuristic function of Graphplan produces poor estimates.

Figure 3.4 shows a planning graph for the STRIPS model in Figure 3.3. The graph is expanded to two time steps to find a goal. All objects in the graph possess

[2]Consider the Gripper domain for example, where there is a robot with N grippers that moves in a space, composed of K rooms that are all connected to each other. All the rooms are modeled as points with connections between each pair of points and therefore the robot is able to reach all rooms starting from any one of them with a simple movement. In the Gripper domain there are L numbered balls which the robot must carry from their initial position to their destination.

their situation. For instance, w_2^1 means doing the `findRestaurant` action in the first step. The procedure to expand from the left to the right graph in Figure 3.4 is as follows:

- Level 0 starts with the initial state of A, B, and C.
- Level 1 consists of possible actions that have preconditions satisfied from Level 0. Action w_2^1 is possible due to A and B. Note that there are three "maintenance actions" for A, B, and C, respectively. A maintenance action is called as "no-op."
- Level 2 consists of the possible effects from the actions in Level 1. A, B, and C are possible due to maintenance actions. D becomes possible from action w_1^2.
- Level 3 contains all actions from Level 1 and additional actions. Action w_3^1 becomes possible due to D which is added at Level 2.
- Level 4 consists of all possible effects from the actions in Level 3. E becomes possible because of action w_3^1. Finally, the goal requirement "direction" is satisfied ("direction" becomes known). Graphplan then proceeds to search backward to find a valid plan as shown in the right graph in Figure 3.4.

3.3.2 SATPlan based reduction

The planning graph on the left side of Figure 3.4 can be converted into a set of logical statements [55]. First, the initial state at the zero can be expressed as: $A^0 \wedge B^0 \wedge C^0 \wedge \neq D^0 \wedge \neq E^0$. We also describe the goal states at the latest level as E^4. Then, we can describe the relations between actions and their preconditions as follows:

$w_2^1 \rightarrow A^0 \wedge B^0$, $Keep(A^1) \rightarrow A^0$, $Keep(B^1) \rightarrow B^0$, $Keep(C^1) \rightarrow C^0$, $W_2^3 \rightarrow A^2 \wedge B^2$, $Keep(A^3) \rightarrow A^2$, $Keep(B^3) \rightarrow B^2$, $w_1^3 \rightarrow C^2 \wedge D^2$, $Keep(C^3) \rightarrow C^2$, $Keep(D^3) \rightarrow D^2$.

where $Keep\{Actionname\}$ corresponds to the maintenance action (i.e., no-op) in Graphplan. In addition, we can express the inference relations between each fact and all previous actions that result in the fact as follows:

$A^4 \rightarrow KeepA^3$, $B^4 \rightarrow KeepB^3$, $C^4 \rightarrow KeepC^3$, $D^4 \rightarrow w_2^3 \vee KeepD^3$, $E^4 \rightarrow w_1^3$, $A^2 \rightarrow KeepA^1$, $B^2 \rightarrow KeepB^1$, $C^2 \rightarrow KeepC^1$, $D^2 \rightarrow w_2^1$.

Finally, these logical statements are combined into one conjunction which has a form of the satisfiability problem (SAT). As mentioned before, the SAT problem can be solved by any off-the-shelf tools (e.g., complete methods including Truth Table and Resolution, or incomplete methods including WalkSat). The final solution of the problem is: $A^0 \wedge B^0 \wedge C^0 \wedge w_2^1 \wedge KeepC^1 \wedge C^2 \wedge D^2 \wedge w_1^3 \wedge E^4$.

3.3.3 Integer Linear Programming (ILP) formulation

The planning graph in Figure 3.4 can also be formulated as a set of constraints. For brevity, suppose that Levels 0 and 1 are period 1, Level 2 and 3 are period 2, and Level 4 is period 3. Then,

- Variables:
  - $X_{e,i}$: if effect e is true in period i, then 1. Otherwise 0.
  - $Y_{a,i}$: if action a is carried out in period i, then 1. Otherwise 0.
  - $Y_{e,i}$: the maintenance action for effect e during period i.
- Objective function:
  $\min \sum_{(a,j) \in action\ set} Y_{a,j} + \sum_{(e,i) \in maintenance\ set} Y_{e,i}$
- Initial constraints:
  $X_{A,1} = X_{B,1} = X_{C,1} = 1$, $X_{D,1} = X_{E,1} = 0$
- Goal constraint:
  $X_{E,3} = 1$
- Constraints for action preconditions:
  $Y_{w_2,1} \leq X_{A,1}$, $Y_{w_2,1} \leq X_{B,1}$, $Y_{A,1} \leq X_{A,1}$, $Y_{B,1} \leq X_{B,1}$, $Y_{C,1} \leq X_{C,1}$, $Y_{w_2,2} \leq X_{A,2}$, $Y_{w_2,2} \leq X_{B,2}$, $Y_{A,2} \leq X_{A,2}$, $Y_{B,2} \leq X_{B,2}$, $Y_{w_1,2} \leq X_{D,2}$, $Y_{C,2} \leq X_{C,2}$, $Y_{D,2} \leq X_{D,2}$
- Backward constraints:
  $X_{A,3} \leq Y_{A,2}$, $X_{B,3} \leq Y_{B,2}$, $X_{C,3} \leq Y_{C,2}$, $X_{D,3} \leq Y_{w_2,2} + Y_{D,2}$, $X_{E,3} \leq Y_{w_1,2}$, $X_{A,2} \leq Y_{A,1}$, $X_{B,2} \leq Y_{B,1}$, $X_{C,2} \leq Y_{C,1}$, $X_{D,2} \leq Y_{w_2,1}$

The objective function is to minimize the number of actions in this program. In order to get an optimal solution for the ILP model above, one can use any integer linear programming solver. In this example, an optimal solution is 3 with ($Y_{w_2,1} = 1$, $Y_{w_1,2} = 1$, $Y_{C,1} = 1$), which is identical to the solutions obtained in Graphplan and SATPlan formulations.

Both Graphplan and ILP are suitable for smaller planning problem with complex operators. On the other hand, SATPlan can be used to find sub-optimal compositions for a large-scale problem with complex operators. While Graphplan and SATPlan can address only the shortest time step to reach a goal, ILP can formulate the objective function so that diverse Quality of Service (QoS) (e.g., response time, service cost, or availability of sources) can be optimized.

The methods reviewed in this chapter have limitations on addressing WSC problems in terms of computational efficiency and effectiveness as follows:

- Graphplan and SATPlan: Both AI planners are all optimal parallel planners that minimize the number of time steps, but not necessarily the number of actions (i.e., the number of web services). It is because both consider WSC problem to be decision problems, not optimization problems. As a result, both search strategy can produce poor solutions in the WSC problem domain, yielding low effectiveness.

- ILP: Even a simple SAT problem can generate a significantly large ILP problem. Obviously, it leads to a significantly long response time, suggesting poor computational efficiency.

WSPR, introduced in Chapter 6, will address these limitations by introducing a two-step search scheme for computational efficiency and a novel heuristic for effectiveness.

In Chapter 4, we will explore public web services and the ICEBE05 test sets, and study their properties from the complex network viewpoint.

Chapter 4

Study of Existing Web Services

In this chapter, we study public web services and the ICEBE05 test sets, and investigate their network features by using complex networks as a basis. For this purpose, we first define the web-service networks.

A set of web services forms a network (or directed graph). There are different kinds of models to determine nodes and edges of the network depending on

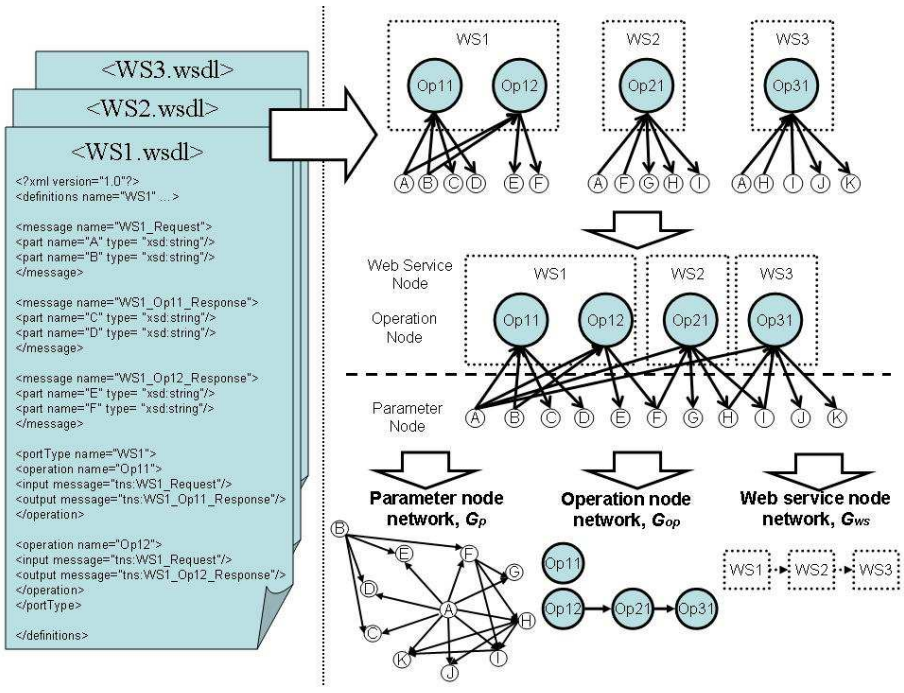


Figure 4.1: Web-service networks

the granularity level: web-service level (coarse granularity), operation level, and parameter level (fine granularity) models. Figure 4.1 illustrates that three WSDL files can be converted into a bipartite graph structure that consists of three distinct kinds of vertices (parameter, operation, and web-service node) and directed arcs between bipartite nodes (operation nodes and parameter nodes). An edge incident from a parameter node to an operation node suggests that the parameter is one of the inputs of the corresponding operation. Conversely, an edge incident from an operation node to a parameter node implies that the parameter is one of the outputs of the corresponding operation. The graph in Figure 4.1 has three web services, labeled WS1, WS2, and WS3. WS1 has two operations, Op11 and Op12. WS2 and WS3 have one operation, Op21 and Op31, respectively. The graph also displays eleven parameters, labeled A through K. According to the node granularity, we can project the upper graph into three different web-service networks.

Definition 4.0.1 (Parameter node network)**.** *A parameter node network is a directed graph, $G_p(V_p, E_p)$, in which V_p is a set of all parameter nodes and E_p is a set of directed edges from input parameters $p_i \in V_p$ to output parameters $p_j \in V_p$; i.e., there exists an operation that has an input parameter matching p_i and an output parameter matching p_j.*

Definition 4.0.2 (Operation node network)**.** *An operation node network is a directed graph $G_{op}(V_{op}, E_{op})$, in which V_{op} is a set of all operation nodes and E_{op} is a set of directed edges from operation $op_i \in V_{op}$ to operation $op_j \in V_{op}$; i.e., op_i can fully or partially match op_j.*

Definition 4.0.3 (Web-service node network)**.** *A web-service node network is a directed graph $G_{ws}(V_{ws}, E_{ws})$, in which V_{ws} is a set of all web-service nodes and E_{ws} is a set of directed edges from web-service node $ws_i \in V_{ws}$ to $ws_j \in V_{ws}$; i.e., there exists one or more edges between any operation in ws_i and any operation in ws_j in a operation node network.*

For example, in Figure 4.1, $A \rightarrow Op11 \rightarrow C$ is projected into $A \rightarrow C$ in the parameter node network, G_p. Similarly, since Op12 partially matches Op21 and subsequently, Op21 partially matches Op31, Op12→Op21→Op31 is shown in the operation node network, G_{op}. In addition, since WS1 possesses Op12, and

WS2 possesses Op21, WS1→WS2 appears in the web-service node network, G_{ws}. As mentioned previously, in operation node networks, a directed arc, $(i, j) \in E_{op}$ suggests that $i \in V_{op}$ can match $j \in V_{op}$ either fully or partially. We can also define a new type of operation node network by restricting the partial-match and only allowing the full-match. We differentiate this operation node network by using the f symbol: $G^f_{op}(V^f_{op}, E^f_{op})$.

To investigate the properties of public web services, we first downloaded 1,544 raw WSDL files[1] that Fan et al. [41] gathered from real-world web services registries such as Bindingpoint [17], Salcentral [96], WebserviceList [98], WebserviceX [120], and xMethod [128]. After weeding out invalid WSDL files that do not conform to the WSDL DTD, 670 valid WSDL files remained. Then, we converted semantically invalid parameters such that those parameters can capture their underlying semantics when they have evident semantic sources. For example, many operations of WSDL files have terms like "result(s)" or "return(s)" in their output parameters which make it hard to interpret what the terms mean; i.e., what the "results" or "returns" means. In this case, we replaced the terms with their operation name or web service name. Sometimes, an operation name is a sequence of concatenated words, therefore hard to do the replacement tasks. In such a case, we did a proper token segmentation and extracted all primal terms from the tokens using lexical analysis. For example, if an operation name is "getAuthornamefromPaper" or "searchAuthornamesbyPaper", we extracted "Authornames" and used it to replace "result(s)" or "returns(s)". After these preprocessing tasks, we built a set, P={all parameters found in the valid 670 WSDL files}, and then we measured the parameter usage and its distribution for $\forall p \in P$. In Section 4.1, we will discuss the parameter usage and its distribution by illustrating a couple of real examples.

4.1 Parameter Usage Distributions

In this section, we build parameter usage distributions of publicly available web services. We define the parameter usage as the frequency of the parameter in the corresponding web service repository, and denote it by $\#(p)$, where $\forall p \in P$. For a better understanding of $\#(p)$ and the $\#(p)$ distribution, we illustrate the

[1] http://rakaposhi.eas.asu.edu/PublicWebServices.zip

distribution of existing public web services, as shown in Figure 4.2. In the figure, the x-axis represents $\#(p)$ and the y-axis represents the number of parameters with same $\#(p)$. The distribution has no humps. We also plotted a power-function, $P_w(\#(p)) \propto \#(p)^{(-\gamma)}$ over the $\#(p)$ distribution, and obtained that the exponent γ is 1.1394. Although 1.1394 is not sufficient to assert that the distribution follows the power law [32], the distribution is highly skewed enough to be seen as a Zipf[2] distribution. Indeed, the parameters such as "license key", "start date", "end date" or "password" have a large $\#(p)$, but most parameters appear just once. This observation implies the existence of hub parameters[3], which appear in web services frequently and serve an important role in the interconnections between web services. For example, since "license key" is such a hub parameter, if we develop a web service which can manage people's license keys in highly secured manner, then we can obtain a large number of connections from many people or other web services.

Similarly, we generated the parameter usage distributions and plotted the power-function using the test sets provided by EEE05 and ICEBE05. As shown in Figure 4.3(a), the test set used in EEE05 has a highly skewed Zipf distribution without any hump. Interestingly, $\gamma = 1.3012$ of EEE05 is similar to public web services, where γ is 1.1394. As a whole, the EEE05 test set can be used to approximate public web services in terms of its $\#(p)$ distribution shape. However, the size of the EEE05 (the number of WSDL files) is only 100 and it is too small to represent the current state or future evolution of web services. For revealing the difference between the EEE05 test set and public web services, we can conduct a statistical analysis on $\#(p)$ results data. Remember that two sets are different in terms of their sizes. Therefore, we use two-sample T-test. This analysis showed

[2]George Kingsley Zipf(1902-1950), was an American linguist who studied statistical occurrences in different languages. He is the eponym of Zipf's law and Zipf-shaped distribution. The Zipf's law states that while only a few words are used very often, many or most are used rarely, resulting in $P_n \sim 1/n^a$, where P_n is the frequency of a word ranked n^{th} and a is almost 1. This means that the second item occurs approximately 1/2 as often as the first, and the third item is 1/3 as often as the first, and so on.

[3]In a network with a scale-free degree distribution, some nodes have a degree that is orders of magnitude larger than the average - these nodes are often called "hubs". The existence of hub nodes is the most notable characteristic in a scale-free network. The hub nodes are thought to serve specific purposes in their networks, although this depends greatly on the domain.

significant differences between two sets (t value[4]=-1.877 < -t(0.95;∞)[5]). The low t value indicates a strong probability that one sample's mean and variance are significantly different from another, because t value implies the probability that public web services and the EEE05 test set are samples from populations with the same mean and variance. From this analysis, we can conclude that the EEE05 test sets cannot approximate public web services.

Regarding the ICEBE05 test sets, we can only demonstrate three distributions for 100-4, 100-16, and 100-32 of Composition-2 due to the lack of space footnoteSee Appendix B for more detailed information on the ICEBE05 test sets.. However, we can assert that the rest of the distributions have a similar shape as the other three distributions. As opposed to the EEE05, all ICEBE05 test sets have $\gamma \leq 0.5$ and four equal humps. For example, Figure 4.3(d) shows four humps at around 1, 100, 200, and 800 $\#(p)$ with the highest hump at 200. The shape of this distribution differs considerably from the EEE05 and public web services. However, before we proceed to point out that the assumptions to build those test sets are conflicting with the state of existing public web services, one thing should be noted. That is, many researchers claim that most applications of web services are likely to be in intra-corporate scenarios rather than on the public web. Under the intra-corporate scenarios, it is questionable as to whether their $\#(p)$ still follow Zipf-like distributions. It is likely to be either a bell-shape or a uniform-shape distribution, because in general, intra-corporate web services tend to be access-restricted or well designed. This can avoid such extreme hub parameters because of load balancing and efficiency problems or network survivability issues. However, even though we allow exceptions by acknowledging the existence of different web-service applications, we still find the appearance of $\#(p)$ distributions with four humps of ICEBE05 to be too artificial.

---

[4]The formula for the two-sample T-test is $t = \frac{\bar{X}_{public} - \bar{X}_{EEE}}{\sqrt{\frac{Var_{public}}{n_{EEE}} + \frac{Var_{EEE}}{n_{EEE}}}}$. $\bar{X}_{public}$ denotes the average of $\#(p)$ of public web services and $\bar{X}_{EEE}$ denotes the average of $\#(p)$ of public the EEE05 test set. Var_{public} and Var_{EEE} denote the variance of $\#(p)$ of public web services and the variance of the EEE05 test set, respectively. n_{public} and n_{EEE} represent each sample size of public web services and the EEE05 test set.

[5]-t(0.95;∞) = -1.645

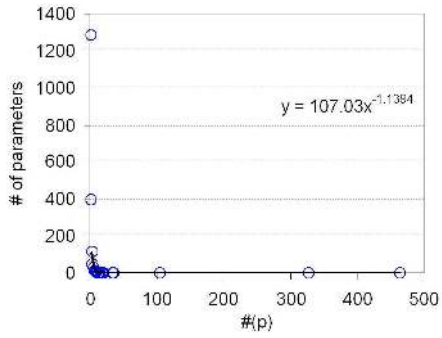
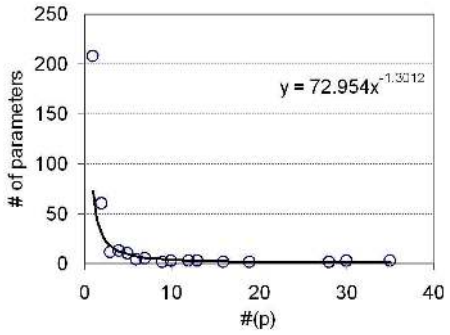
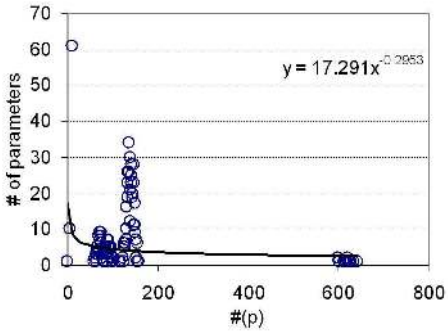


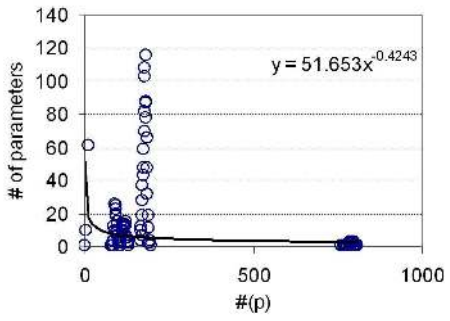
Figure 4.2: Parameter usage distribution for public web services



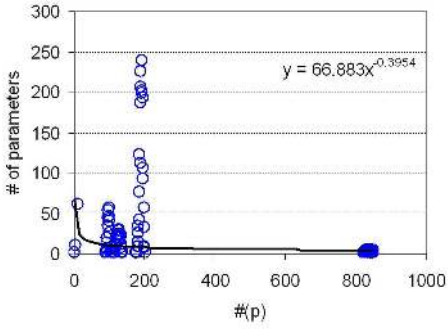
(a) EEE05



(b) 100-4 in Composition2 of ICEBE05



(c) 100-16 in composition2 of ICEBE05



(d) 100-32 in composition2 of ICEBE05

Figure 4.3: Parameter usage distribution for EEE05 and ICEBE05

4.2 Random and Complex Network Models

A network is usually defined as a set of nodes and links. The nodes represent entities, such as persons, machines, molecules, documents, or businesses; the links represent relationships between pairs of entities. A link can be directed (one-way relationship) or undirected (mutual relationship). A hop is a transition from

one node to another across a single link separating them. A path is a series of hops. Networks are very general and can represent any kind of relation among entities [32]. There are many systems to form complex networks, in which vertices signify the elements of the system and edges represent the interactions between them as follows:

(1) Living systems form a huge genetic network, in which vertices are proteins and genes, the chemical interactions between them represent edges [123].

(2) At a different organizational level, a large network is formed by the nervous system, whose vertices are the nerve cells, connected by axons [1].

(3) In social sciences, complex networks occur where vertices are individuals and the edges are social interactions between them [117].

(4) In the World Wide Web (WWW), vertices are HTML documents connected by links pointing from one page to another [4].

A network topology refers to interconnection patterns between nodes of the network. Some common network topologies are:

- Clique or island: a connected sub-network may be isolated from other cliques.
- Hierarchical network: a network is connected as a tree.
- Hub and spoke: there is a special node, the hub, which connects to every other node directly.
- Multi-hub network: there are several hubs connected directly to many nodes.

Some network topologies are planned, such as the electric grid, the interstate highway system, or the air traffic system while many others are unplanned, such as the Internet.

We implemented a novel benchmark tool named WSBen[6] that generate a collection of synthetic web services (WSDL) files to test WSD and WSC algorithms. WSBen provides three network models so that users can specify the underlying network topology of their test sets (WSDL files). The three network models are

[6]WSBen will be discussed in Chapters 5, Appendix A, and Appendix E in detail.

random graph model, small-world network model and scale-free network model. We will describe the three networks with focus on their network building algorithms in the following sections.

4.2.1 Random graph model

Definition 4.2.1 (Random graph)**.** *$Rd_{(N,p)}$ is defined as a random graph on N nodes, if each pair of nodes is connected with probability p. As a result, edges are randomly placed among a fixed set of nodes. A random network can be constructed by means of the Erdos-Renyi's random-graph model [38].*

A random graph model [38] is built by a simple and straightforward procedure, where we start with N vertices and connect each pair of vertices with probability p. As a result, links are randomly placed among a fixed set of nodes. The probability distribution that a node has v number of edges follows a Poisson distribution, $P(v) = \frac{e^{-\lambda}\lambda^v}{v!}$, where $\lambda = N\binom{N-1}{v}p^v(1-p)^{N-1-v}$. When the average degree of a node is less than 1, there is a high probability that the network is a set of disconnected islands. Conversely if the average is 1 or greater, the chance that the entire network is connected, increases considerably. For many years, this model had been used to approximate real networks. However, when researchers started measuring the connection distribution of real networks, they found that the actual distributions do not match this random graph model. Instead, they found that real distributions tend to follow a power-law distribution [32].

4.2.2 Small-world network model

Definition 4.2.2 (Regular Network)**.** *$R_{(N,k)}$ is defined as a regular network on N nodes, if node i is adjacent to nodes $[(i+j)\ mod\ N]$ and $[(i-j)\ mod\ N]$ for $1 \leq j \leq k$. If $k = n$, $R_{(N,k)}$ becomes a complete N-nodes graph, where every node is adjacent to all the other $N-1$ nodes.*

Definition 4.2.3 (Small-world network)**.** *Small-world networks are characterized by a highly clustered topology like regular lattices and small average shortest distance between nodes. Both highly clustered structure and small world property are referred as the small-world network properties [31].*

By using the Watts-Strogatz model [118, 119], we can construct networks that have small-world network properties. The model depends on two parameters, connectivity (k) and randomness (p), given the desired size of the graph (N). The Watts-Strogatz model starts with a $R_{(N,k)}$ and then every edge is rewired at random with probability p; for every edge (i, j), we decide whether we change j node (the destination node of (i, j)) with probability p. The Watts-Strogatz model leads to different graphs according to the different p as follows:

- When $p = 0$, an $R_{(N,k)}$ graph is built.
- When $p = 1$, a completely random graph is built.
- Otherwise, with $0 < p < 1$, each edge (i, j) is reconnected with probability p to a new node k that is chosen at random (no self-links allowed). If the new edge (i, k) is added, then (i, j) is removed from the graph. The long-range connections (short-cuts) generated by this process decrease the distance between the vertices. For intermediate values of p, there is the "small-world" region, where the graph is highly clustered and has a small average path length.

We can define some metrics to quantify the characteristic properties of the small-world network as follows:

- L: the average shortest distance[7] between reachable pairs of vertices. $L(p)$ is defined as L of the randomly rewired Watts-Strogatz graph [119] with probability p. L_{random} is identical to $L(1)$.
- C: the average clustering coefficient. Suppose that for a node i with v_i neighbors, $C_i = \frac{2E_i}{v_i(v_i - 1)}$, 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. $C(p)$ is defined as C of the randomly rewired Watts-Strogatz graph with probability p. C_{random} is identical to $C(1)$.

WSBen can support the Newman-Watts-Strogatz model [77] as well as the Watts-Strogatz model [119]. The difference between these two models is that the

[7]The distance between nodes refers to the number of hops between the nodes.

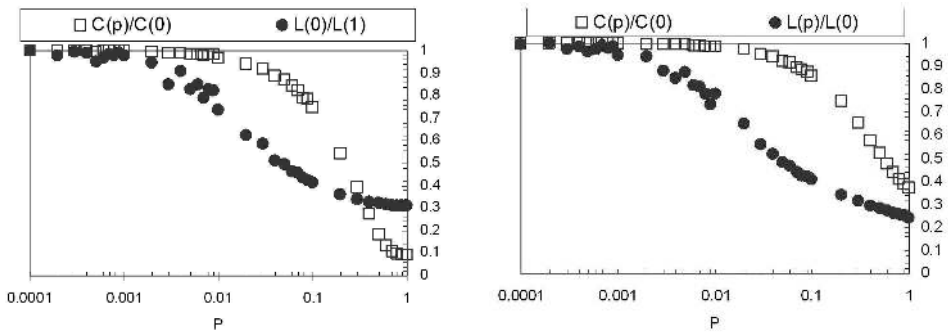


Figure 4.4: Characteristic path length $L(p)$ and clustering coefficient $C(p)$ for the family of randomly rewired graphs. (left) Watts-Strogatz model. (right) Newman-Watts-Strogatz model

Newman-Watts-Strogatz model adds new edges (shortcuts) without removing old edges while the Newman-Watts model does not. Figure 4.4 shows $L(p)$ and $C(p)$ for the family of randomly rewired graphs generated by both the Newman-Watts-Strogatz model and the Watts-Strogatz model, where $N = 100$ and $k = 6$. The data shown in the figure are averaged over 20 random realizations of the rewiring process. As shown in the figure, for intermediate values of p, there is the "small-world" region, where the graph is highly clustered yet has a small average path length. Notice that Newman-Watts-Strogatz model does not result in a completely random graph, even when $p = 1$ because this model keeps $R_{(N,k)}$ until the graph generation procedure ends, keeping in mind that Newman-Watts-Strogatz model does not remove any edges during its rewiring process. In the next section, we will discuss the scale-free network model that was proposed to approximate real networks characterized by the power-law distribution.

4.2.3 Scale-free network model

Definition 4.2.4 (Scale-free network)**.** *Networks are called scale-free networks if the number of nodes that have v neighbor nodes is proportional to $P_w(v) \propto v^{(-\gamma)}$, where γ is typically greater than two with no humps.*

It is generally accepted that networks with power-law connection statistics are called scale-free networks [32]. A power-law distribution has no humps with the power p which is greater than two. For these reasons, the connection distribution

of the scale-free networks are compatible with a power-law with the exponent of two.

Barabasi and Albert [3, 4, 31] have recently proposed a set of different models to build scale-free graphs using the base conditions as follows:

- Growth: new nodes appear at random times,
- Preferential attachment: a new node connects to an existing node with probability proportional to the number of connections already at the node.

WSBen uses the Barabasi and Albert extended model to provide the scale-free properties. The extended model uses an algorithm to build graphs that depend on four parameters: m_0 (initial number of nodes), m (number of links added and/or rewired at every step of the algorithm), p (probability of adding links), q (probability of edge rewiring). The procedure starts with m_0 isolated nodes and performs one of the following three actions at every step:

(1) With the probability of p, $m(\leq m_0)$ new links are added. The two nodes are picked randomly. The starting point of the link is chosen uniformly, and the end point of the new link is chosen according to the following probability distribution:

$$\Pi_i = \frac{v_i + 1}{\sum_j (v_j + 1)} \tag{4.1}$$

where Π_i is the probability of selecting the ith node, and v_i is the degree of node i. The process is repeated m times.

- With the probability of q, m edges are rewired. For this purpose, i node and its link l_{ij} are chosen at random. The link is deleted. Instead, another node z is selected according to the probabilities of 4.1, and the new link l_{iz} is added.
- With the probability of $1 - p - q$, a new node with m links is added. These new links connect the new node to m other nodes chosen according to the probabilities of 4.1.

Once the desired number N nodes are obtained, the algorithm stops. The graphs generated by this algorithm are scale-free graphs, and the edges of the graphs are constructed such that the correlations among edges do not form. When $p = q$, the algorithm results in a graph, whose connectivity distribution can be approximated by

$$P(v) \propto (v+1)^{-\frac{2m(1-p)+1-2p}{m}+1} \tag{4.2}$$

where v is the degree of a node. Note that the current implementation of WSBen sets $m_0 = m$ and $p = q = 0$ as the default configuration so that it can easily generate simplified scale-free graphs.

4.2.4 Summary

Networks can be called small-world networks if they share the interesting properties of both random and regular networks: highly clustered and small network diameter ℓ. Specially, $C \gg C_{random}$ and $L \gtrsim L_{random}$, or $\ell \approx log(N)$, where N is the size of a network. Similarly, networks can be called scale-free networks if their γ is greater than 2 with no humps, where γ is the exponent of a power function, $P_w(v)$ (the number of nodes, that have v number of neighbor nodes) that has the form of $P_w(v) \propto v^{(-\gamma)}$. It is generally accepted that networks with power-law connection statistics are called scale-free networks. A power-law distribution has no humps with the power p greater than two. For these reasons, we characterize the scale-free networks as networks with their γ greater than two and with no humps.

4.3 Real Web-service Networks

In this section, we will build web-service networks (G_p, G_{op}, G_{op}^f and G_{ws}) of public web services and ICEBE05 test sets, and measure their scale-free and small-world network properties.

Table 4.1: Features of public web-service networks

Features	G_p	G_{op}	G_{op}^f	G_{ws}
# of nodes	4,456	991	293	273
# of arcs	10,728	3,796	1,145	752
Network Diameter	8	7	4	9

4.3.1 Public web services

Table 4.1 shows the general information about each of public web-service networks. We found that G_{op}^f has the longest shortest path with 4 hops. From a data mining perspective, we can discover what web services are on that path. The path is a chain of five web services connected by "fully-matching" operations as shown in Figure 4.5. The five web services commonly provide data conversion related services. For example, the second web service in the path, "`Japanese zipcode To address Converter`", inputs "Address" and finds the "Zipcode" corresponding to the "Address".

This discovery however has a fundamental problem in its underlying matching assumption. Note that we assumed only syntactic (text-based) matching among web services, such that if two parameters have the same spellings, they are matched. For example, "List," the output parameter of "`LocInfo Zip Code`" web service can simply match "List," the input parameter of "`Online Software Shop`" web service. However, it is evident that both parameters have different semantics, as the former "List" means the list of zip codes but the latter "List" indicates the list of software. Thus, from the semantic view, the last web service must be removed from the path resulting in leaving only four web services in the path. Subsequently, we find that there are no compositions with more than four web services linked through fully-matching relations. This observation points out that we are still in the formative stage, as far as publicly available web services are concerned in the sense that it is not easy to discover correlations between public web services due to the sparse population of web services.

We can investigate the scale-free network properties of public web services. Figure 4.6, 4.7, 4.8 and 4.9 show G_p, G_{op}, G_{op}^f and G_{ws} of public web services with their corresponding outgoing edge distributions. In the outgoing edge distributions, the x-axis represents the outgoing edge degree and the y-axis represents the number

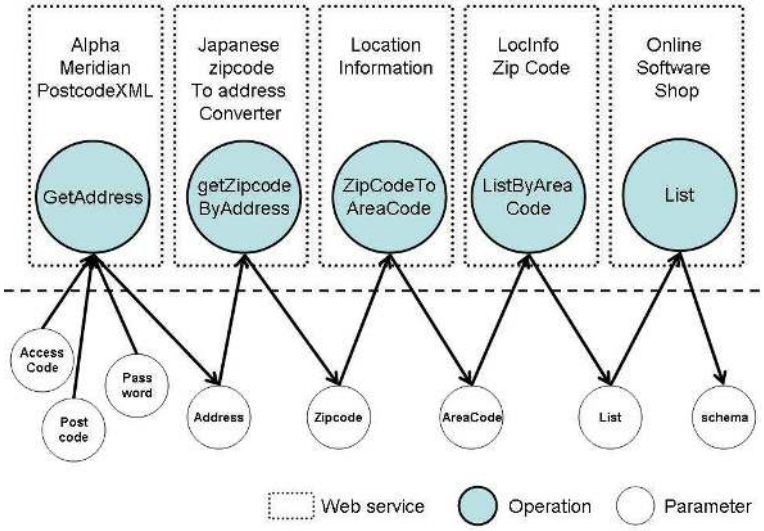


Figure 4.5: Web services on the network diameter path in G_{op}^{f}

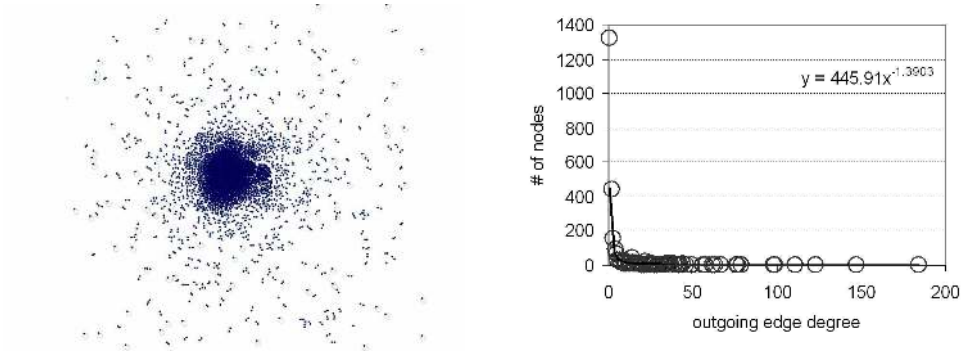


Figure 4.6: G_p of public web services. (left) G_p. (right) outgoing edge distribution of G_p

Table 4.2: Scale-free network properties

	G_p	G_{op}	G_{op}^{f}	G_{ws}
γ	1.3903	1.1152	1.5775	1.3073

of nodes with the outgoing edge degree. We can apply the power function, $P_w(v)$, to each of the outgoing edge distributions and check γ, the exponent value of $P_w(v)$.

Even though all γ values do not exceed 2, as shown in Table 4.2, they are highly skewed so that their shape is similar to the Zipf distribution. Note that the x-axis of the original Zipf distribution uses the rank of the frequency of a word

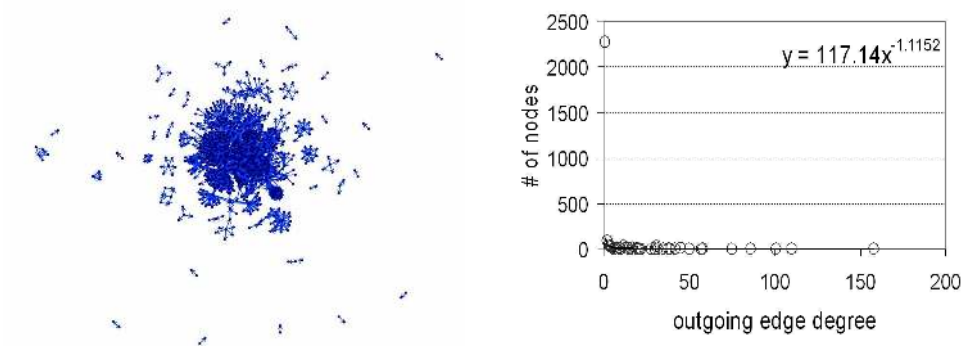


Figure 4.7: G_{op} of public web services. (left) G_{op}. (right) outgoing edge distribution of G_{op}

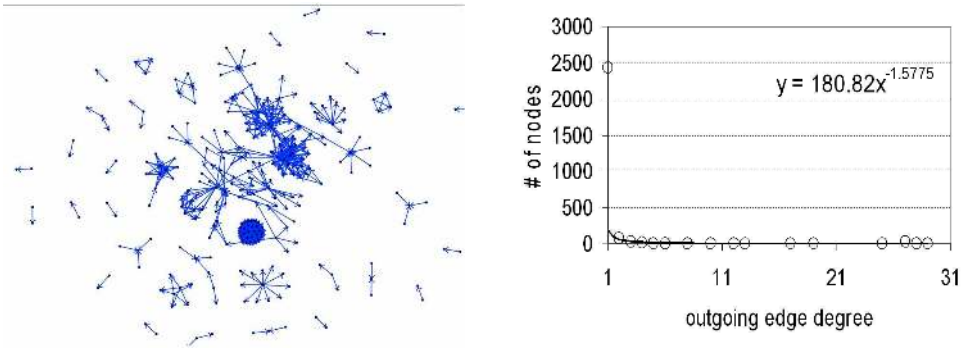


Figure 4.8: G^f_{op} of public web services. (left) G^f_{op}. (right) outgoing edge distribution of G^f_{op}

in a specific language which is different from our distribution where the x-axis suggests the degree of outgoing edges. This observation implies the existence of hub nodes, such as hub parameters, hub operations and hub web services. If a parameter is a hub node in $G_p(V_p, E_p)$, the node has a large incoming edge degree or outgoing edge degree, compared with other nodes. Hub operations or hub web services can be defined in the same way as hub parameters. From the data mining perspective, it is interesting to identify the hub nodes in each network. For example, in G_p, the parameter with the largest outgoing edge degree (=184) is "password". This implies that "password" is the most frequently used input parameter in the context of public web services. On the other hand, in G_{ws}, the web service with the largest outgoing edge degree is "`Amazon Web Services 2.0`" with an outgoing edge degree of 59. Indeed, the web service contains as many as

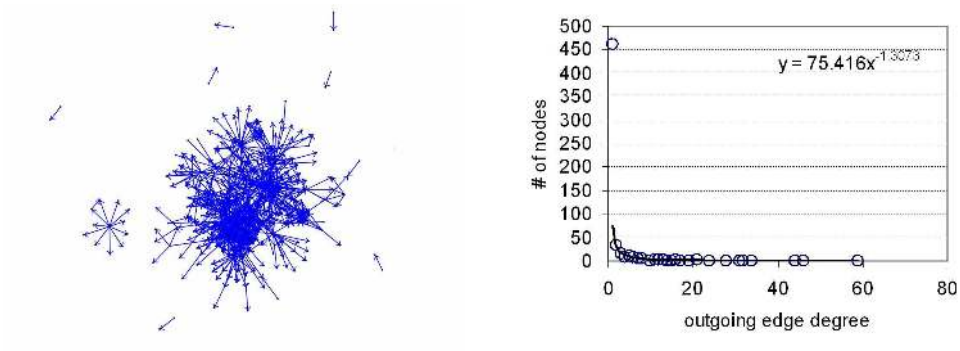


Figure 4.9: G_{ws} of public web services. (left) G_{ws}. (right) outgoing edge distribution of G_{ws}

Table 4.3: Small-world network properties of giant components in the public web services

Networks	L_{actual}	L_{random}	C_{actual}	C_{random}
G_{op}^{f}	3.4984	2.9029	0.1458	0.0767
G_{p}	5.6144	4.6918	0.0451	0.0009
G_{op}	4.4997	3.5313	0.3064	0.0099
G_{ws}	3.35	3.2971	0.2753	0.0256

23 operations, which enable vast and diverse correlations with other web services.

We can investigate the small-world network properties of public web services. Table 4.3 shows the average path length, L and clustering coefficient, C for four giant connected components extracted from each of the four public web-service networks (G_{ws}, G_p, G_{op}, and G_{op}^{f}), compared to random graphs with the same number of nodes and average degree of a node. Note that we treat all edges of each network of G_{ws}, G_p, G_{op}, and G_{op}^{f} as undirected and un-weighted, recognizing that these are crude approximations. However, this approach is acceptable in the complex network analysis community. For example, Watts and Strogatz considered the electrical power grid of the western United States [86] as an undirected and un-weighted graph in their paper [119]. In practice, high-voltage transmission lines must be directed, starting from generators to substations through transformers. However, when investigating the small-world network properties of a given network, we usually restrict our attention to the network topology in itself rather than the flow of materials on edges (e.g., information, electricity). Another reason to ignore the directions of edges has to do with measuring the average path length, L that

Table 4.4: Summary of giant components in public web services

Networks	# of nodes(A)	# of nodes in giant component(B)	$\frac{B}{A} \times 100(\%)$	average # of edges per node in giant component
G_{op}^{f}	293	89	30.3(%)	5.21
G_{p}	4,456	3,117	69.9(%)	5.94
G_{op}	991	835	84.2(%)	7.71
G_{ws}	273	247	90.4(%)	5.93

is averaged over the shortest path between reachable nodes. The problematic situation arises when each of two networks (one of the web-service networks and its random network pair) has a different number of reachable nodes. It is clear that if the number of reachable nodes is different, it is inconsistent to simply compare two L values. On the contrary, if we assume that both networks are undirected, the number of reachable nodes in both networks become the same, because there is no un-reachable node pair in each of the networks.

Similar to previous works on small-world network [119], we restrict our attention to the giant connected components [20]. The information about giant components of each web-service network are summarized in Table 4.4.

It is an interesting result that G_p, G_{op}, and G_{ws} show small-world network properties: $L \gtrsim L_{random}$ and $C \gg C_{random}$. This result suggests that public web services have such 'short-cuts' that connect nodes that would otherwise be much farther apart than L_{random}. Conversely, G_{op}^{f} is shown to be close to a random network: $L \gtrsim L_{random}$ and $C \gtrsim C_{random}$. However, this result is not consistent with the previous observation that G_{op}^{f} has the highly skewed outgoing edge degree distribution. In general, the degree distribution of the random network tends to have a bell shape rather than the skewed shape. This misleading result is a side-effect of removing directions and weights on the edges of G_{op}^{f}. Remember that we treat all edges of each network of G_{ws}, G_p, G_{op}, and G_{op}^{f} as undirected and unweighted in order to measure small-world network properties efficiently. Therefore, in this case, we have to turn our attention to the original G_{op}^{f}, and conclude that G_{op}^{f} is more close to the scale-free network than the random network.

Table 4.5: Features of the ICEBE05 web-service networks

Features	G_p	G_{op}	G_{op}^f
# of nodes	736	1,774	1,229
# of arcs	8,569	31,905	2,599
Network Diameter	7	7	7

4.3.2 ICEBE05 test sets

Similarly, we plotted graphs using synthetic WSDL files that were used in ICEBE05. The ICEBE05 test sets are auto-generated from software by the ICEBE05 organization, and consist of two main parts: Composition-1 and Composition-2. Both portions have nine test sets. The size of 18 test sets is one among 3,356 and 5,356 and 8,356 WSDL files. Each test set has 11 test requests. More detailed information on ICEBE05 is provided in Appendix B. In the interest of space, we show a graph for the first test set named "20-4" of Composition-2, and other cases show a similar pattern. Figure 4.10, 4.11 and 4.12 show G_p, G_{op}, and G_{op}^f separately, with their outgoing edge distributions. The shapes of graph differ considerably from the real public web services. Each graph consists of ten islands, and those islands (the connected component) are uniform in terms of their network topologies (# of nodes, # of arcs and the connectivity pattern). Each island is likely to approximate workflow web-service domains (e.g., business, scientific, and medical workflow). The parameters used in workflows tend to be domain-specific or professional terms, and thus their G_ps are likely to have a regular network property. Moreover, the nodes in G_{op} and G_{op}^f of workflow domains are likely to be connected to a few number of neighboring web services in the succeeding stage of workflow. This occurs because workflows can be constructed such that the underlying networks do not follow scale-free network properties, in order to avoid skewed resource consumption or increase the network survivability.

Table 4.5 shows the general information about each of the ICEBE05 web-service networks. Note that all web services in ICEBE05 have only one operation, meaning that G_{op} and G_{ws} are identical. This is the reason that the network diameters of G_{op} and G_{op}^f are 7 identical. Moreover, this network diameter path is identical to one of the solutions to requests asked by ICEBE05. In fact, all test requests of ICEBE05 can be solved using the full-matching operation alone. In the event that

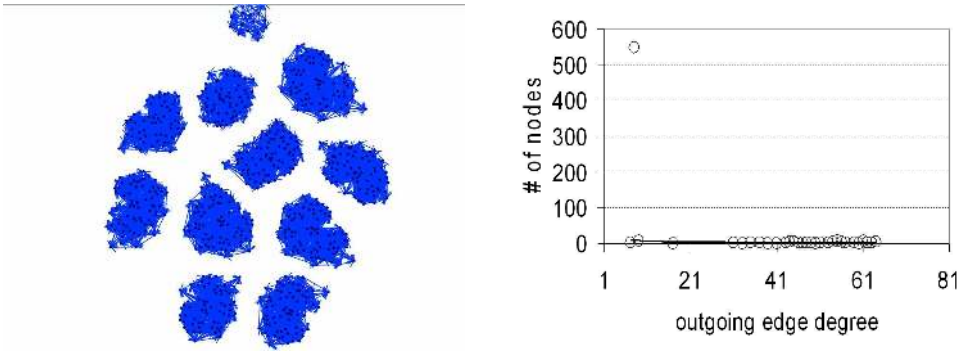


Figure 4.10: G_p of ICEBE05 test set. (left) G_p. (right) outgoing edge distribution

Table 4.6: Small-world network properties of giant connected component in the ICEBE05 web services

Networks	L_{actual}	L_{random}	C_{actual}	C_{random}
G_p	1.7689	1.6905	0.3188	0.3055
G_{op}	3.1613	1.8122	0	0.1883
G_{op}^f	4.8629	3.4552	0	0.0345

the WSC problem has this kind of special scheme, it can be addressed quickly using a simple shortest-path algorithm with a single source. More detailed information about the request complexity of ICEBE05 is provided in Appendix B.

Regarding Figures 4.10 and 4.11, 86% of the nodes of G_p has an outgoing edge degree of 7 uniformly and 77% of the nodes of G_{op} has an outgoing edge degree of 20 uniformly. Hence, the distribution can be fitted by the peak distribution, and this peak shape is the strong evidence to prove that G_p and G_{op} do not follow the scale-free network, which is the case with the public web services (refer Figure 4.2. On the contrary, the outgoing edge distribution of G_{op}^f has $\gamma = 1.5927$, so that it can be regarded as the Zipf-like distribution. However, it is questionable to say that it has scale-free properties, because the largest outgoing degree is just 12. This number is too small to view the node as a hub, considering that the total node number of G_{op}^f is 1,229.

Table 4.6 shows the average path length, L and clustering coefficient, C for the three giant connected components extracted from each of three ICEBE05 web service networks, compared to random graphs with the same number of nodes and

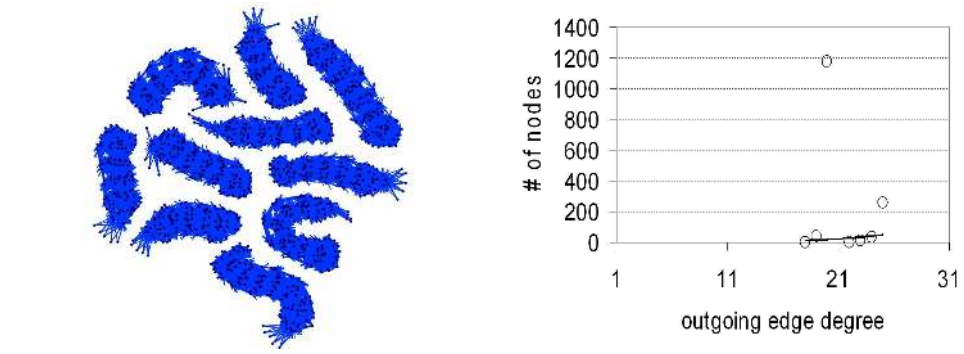


Figure 4.11: G_{op} of ICEBE05 test set. (left) G_{op}. (right) outgoing edge distribution

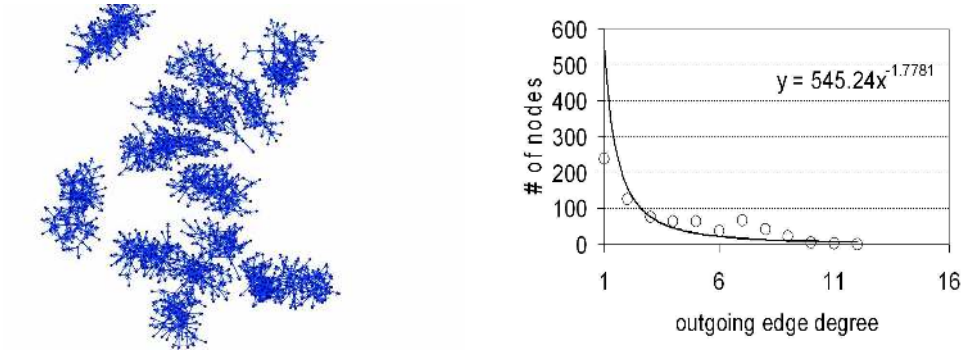


Figure 4.12: G^f_{op} of ICEBE05 test set. (left) G^f_{op}. (right) outgoing edge distribution

Table 4.7: Summary of giant components in ICEBE05

Networks	# of nodes(A)	# of nodes in giant component(B)	$\frac{B}{A} \times 100(\%)$	average # of edges per node in giant component
G_p	736	80	10.8(%)	24.25
G_{op}	1,774	200	11.2(%)	36.7
G^f_{op}	1,229	136	11.06(%)	4.08

average degree of a node. Similar to public web service case, we treat all edges of each network of G_p, G_{op}, and G^f_{op} as undirected and un-weighted, and we restrict our attention to the giant connected component. The information about the giant component of each web-service network is summarized in Table 4.7. G_p shows the

random network characteristics: $L \approx L_{random}$ and $C \approx C_{random}$. On the contrary, G_{op} and G^f_{op} show special characteristics: C=0 and $L \approx L_{random}$. As an extreme case, if a graph is a tree in which no circle exists, then C becomes 0, because there are no triangles in the graph. We can also explain the characteristics by comparing G_{op} and G^f_{op} to typical workflows in the business, medical, and science domains, where each node of G_{op} and G^f_{op} can map into an activity or event of the corresponding workflow. If an activity is connected to a few number of activities in the succeeding stage sequentially, then the first neighbors of the activity are likely to have no connection between them. In that case, the clustering coefficient, C, becomes 0.

4.3.3 Summary

We observe that public web services show similar features as the world wide web, in terms of small-network and scale-free network properties. This is inferred because public web-service networks have grown in a similar manner to that of the Internet. On the contrary, the ICEBE05 web-service networks are planned and highly artificial in the sense that their underlying topologies are the random network or tree structures. Additionally all arcs are directed sequentially like business workflows, and their isolated sub-networks are uniformly partitioned such that the size of sub-networks and the network topology are uniform. This observation generates following implications:

1. WSC problems can arise in diverse scenarios. However, they can be captured by investigating which of the network topologies would fit into their web-service networks, especially using complex networks.

2. We can develop a novel web-service benchmark tool with the capability to generate web services whose underlying network topology is characterized by diverse models.

3. Understanding the structural properties of networks often help gain better insights and develop better algorithms. Therefore, we can make a heuristic search algorithm to address WSC problems by exploiting the underlying network structure. Note that our proposed composition algorithm does not

exploit this network structure but uses a simple strategy in favor of fully-matching web services. This will be discussed in detail in Chapter 6.

4. A web-service network can be relaxed into a parameter node network by ignoring operation and web service information. This relaxation idea can be adopted for building a new web-service composition algorithm. In fact, WSPR conducts a polynomial-time forward search over the relaxed parameter-based search space. This will be discussed in detail in Chapter 6.

In the following chapter, we will present a novel benchmark tool called WSBen, that is designed to reflect the implications learnt from this chapter. We use WSBen to generate diverse test sets which will be later used for testing WSC solutions.

Chapter 5

WSBen: Web Services Discovery and Composition Benchmark Tool

In this chapter, we present a novel benchmark tool titled WSBen[1] to test web-service discovery and composition algorithms. WSBen [80] generates a variety of files to speed up testing processes as follows: (1) A collection of synthetic web services (WSDL) files with diverse characteristics and sizes; (2) Test discovery and composition queries and solutions; and (3) External files for statistical analysis and AI planners. Users can characterize the WSDL files generated using various parameters, such as underlying network model, skewness, and size. To illustrate the application of WSBen, we present a use case of WSBen in the network analysis community, where we attempt to estimate the size of a giant component in WSBen-generated web-service networks using random graph theory.

A web service may have a number of input and output parameters, each possibly instantiated from the same concept. For example, "temperature", "windchill", and "humidity" can be clustered into a single concept, such as `weather`. Web services that have a goal to provide weather information can use parameters in the `weather` cluster. In general, parameters are clustered together based on the following heuristics: *"parameters tend to express a similar concept if they occur together often"*. It is well known that one particular application domain (e.g., travel, reservation, entertainment, and look-up services for diverse areas) can be projected into a cluster, which contains a set of atomic parameters that tend to express the same

[1]See Appendix D for the WSBen manual and other instructions.

concept and occur together with similar frequency [35]. For example, if there is an `address` cluster {"state","city","street","zip" }, then parameters in the `address` occur together with a high frequency. More precisely, we can describe the relationship between parameters in a same cluster, by using the co-occurrence probability $Pr(p_2|p_1)$ that is defined as follows:

$$Pr(p_2|p_1) = \frac{[\#In(p_1, p_2) + \#Out(p_1, p_2)]}{[\#In(p_1) + \#Out(p_1)]} \tag{5.1}$$

where p_1 and p_2 are the parameters. $\#In(p)$ denotes the number of web services that contain a parameter p in their input parameter. Similarly, $\#In(p_1, p_2)$ describes the number of web services that contain both p_1 and p_2 in their input parameter. $\#Out(p)$ and $\#Out(p_1, p_2)$ are defined in the same way. If two parameters p_1 and p_2 are in the same cluster, then p_1 and p_2 have the same co-occurrence probability such that $Pr(p_2|p_1) = Pr(p_1|p_2)$.

Consequently, at a higher level, a web service can be assumed as a transformation between two different application domains, and each can be represented by a cluster. This assumption is the basis in developing WSBen. From the perspective of graph thery, WSBen builds a *Parameter Cluster Network* which consists of clusters and directed arcs connecting two different clusters. These directed arcs become web-service templates from which WSBen generates web services as users specify. Formally, the parameter cluster network is defined as follows:

Definition 5.0.1 (Parameter Cluster Network)**.** *A directed graph $G_{cl}(V_{cl}, E_{cl})$, where V_{cl} is a set of clusters and E_{cl} is a set of directed edges that are incident from input clusters $i \in V_{cl}$ to output clusters $j \in V_{cl}$. Here, cluster i and j contain a set of non-overlapping parameters denoted by Pa_i and Pa_j, respectively, where $Pa_i \cap Pa_j = \emptyset$. Each directed edge is also called a web-service template from which WSDL files are generated.*

In Chapter 4, we demonstrated that web-service networks may have different network topologies by investigating two real cases: the public web and ICEBE05. Thus, we believe that diverse scenarios, including the two cases in addition to many others, need to be evaluated in testing WSD and WSC algorithms. Those diverse scenarios can be generated by varying the parameter cluster network G_{cl}.

There are several graph models we can use as underlying topologies of parameter cluster networks. For example, the Watts-Strogatz [118, 119] or Newman-Watts-Strogatz model [77] can be chosen when a small-world graph is of interest. The Barabasi-Albert [3, 11, 4] model can be used to generate a scale-free graph. WSben supports the view that G_{cl} may be consistent with the complex network models as well as the random graph model, depending on different applications. In the following sections, we introduce several graph models and functions that the current implementation of WSBen supports, as well as real test sets generated by WSBen as illustrative examples.

5.1 Overview of WSBen

The WSBen provides a set of functions to simplify the generation of test environments for WSD and WSC problems. Figure 5.1 shows the overview of WSBen. In detail, WSBen consists of the following functionalities:

- Input framework: users specify and control the generated synthetic WSDL files and their characteristics. WSBen provides two input frameworks: xTS and yTS. They are different from each other in terms of their approaches to specify G_{cl}. xTS applies existing complex and random network models to specify G_{cl}, while yTS allows users to fine-tune implementation options of G_{cl} using diverse distributions. yTS is more flexible than xTS, but experience shows that it requires more effort to handle it fluently due to its flexibility. To avoid the confusion between xTS and yTS, we will treat the two frameworks separately. Therefore, in this chapter, we restrict attention to $xTS = \langle |J|, G_r, \eta, M_p, |W| \rangle$ and each element of xTS will be discussed in more detail below. yTS is further discussed in Appendix A.

- Parameter cluster network, $G_{cl}(V_{cl}, E_{cl})$: If xTS is given by users, based on the first four elements, WSBen generates G_{cl}. Each cluster of G_{cl} is filled with some number of atomic parameters. In this network, web services are defined as transformations between two different clusters. That is, $\langle i, j \rangle \in E_{cl}$ becomes web service templates. The role of web-service templates in the test set generation will be illustrated in Section 5.2.

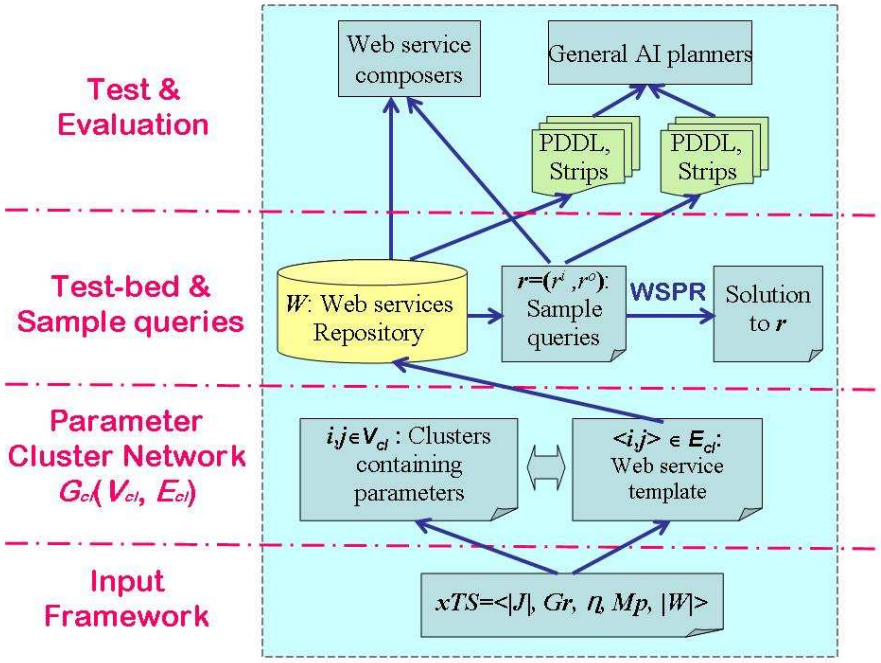


Figure 5.1: Overview of WSBen

- Test set and sample requests: by randomly selecting the web service templates (arcs of the parameter cluster network), WSDL files are generated. Once a test set is generated, users can generate sample test requests $r = \langle r^i, r^o \rangle$, where r^i is a set of atomic parameters contained in the cluster randomly selected from V_{cl}, and r^o consists of the first five largest parameters with $g_{r^i}(p)$[2]. The generation process of test sets and test requests is illustrated in Section 5.2.

- Test and evaluation: it is possible to export both the web service WSDL files and test requests into files in PDDL [69] and STRIPS format, enabling concurrent comparison with state-of-the-art AI planners. Moreover, WSBen can export the parameter usage[3] of all parameters into external files in a comma-separated file format (CSV), enabling users to analyze test sets statistically.

[2] $g_{r^i}(p)$ is the cost of achieving $p \in P$ from r^i by the forward search over the parameter space. A more detailed explanation will be given in Chapters 6 and 7.

[3] The parameter usage is denoted by $\#(p)$. For details, see Section 4.1.

xTS, the 5-tuple framework for WSBen, consists of:
$xTS = \langle |J|, G_r, \eta, M_p, |W| \rangle$. Provided that the first four tuples are grounded, one can build a parameter cluster network, where clusters are nodes and web-service templates are directed edges. Each tuple of xTS is more specifically explained as follows:

(1) $|J|$ is the total number of parameter clusters.

(2) G_r denotes a graph model to specify the underlying topology of a parameter cluster network. G_r can be one of following three models:

- Erdos-Renyi($|J|$, p): as discussed in Section 4.2.1, this model has such a simple generation approach that it chooses each of the possible $\frac{|J|(|J|-1)}{2}$ edges in the graph with $|J|$ nodes with probability p. The resulting graph becomes the same as the binomial graph. Note that the generation of this graph costs $O(|J|^2)$ because it starts with creating $\frac{|J|(|J|-1)}{2}$ edges.
- Newman-Watts-Strogatz($|J|$, k, p): as discussed in Section 4.2.2, the initialization is a regular ring graph with k neighbors. During the generation process, new edges (shortcuts) are added randomly with probability p for each edge. Note that no edges are removed, differing from the Watts-Strogatz model.
- Barabasi-Albert($|J|$, m): as discussed in Section 4.2.3, this graph model is generated by adding new nodes with m edges that are preferentially attached to existing nodes with a high degree. The initialization is a graph with m nodes and no edges. Note that the current implementation of WSBen is limited because it can only generate the simplified version of the extended Barabai-Albert model, as discussed in Section 4.2.3 by setting $p = q = 0$ and $m_0 = m$, resulting in graphs with $\gamma = 2.9 \pm 0.1$, where γ is the exponent of a power function $P_w(v)$ defined over connectivity v range in the form of $P_w(v) \propto v^{-\gamma}$. WSBen will be extended to fully support the features of the extended Barabai-Albert model.

(3) η denotes the parameter condense rate. With η, users can control the number of parameters in produced web services.

(4) M_p denotes the minimum number of parameters a cluster can contain. In other words, clusters may have a different number of parameters but all clusters must have at least M_p number of parameters.

(5) $|W|$ denotes the total number of web services of a test set.

With $|J|$ and G_r, the first two tuples of xTS, we can build G_{cl} with each empty cluster. Thus, we need a procedure to fill each empty cluster with parameters. For this purpose, WSBen uses the following procedure:

(1) A parameter cluster network G_{cl} with empty clusters is built by applying $|J|$ and G_r, the first two tuples of xTS.

(2) Co-occurrence probability of each cluster is measured by the following probability:

$$\Delta_j = \frac{k_j}{\max_{j \in V_{cl}} k_j} \eta \tag{5.2}$$

where Δ_j is the co-occurrence probability of cluster j, and k_j is the edge degree of cluster j. η is the parameter condense rate which is given by users.

(3) $|Pa_j|$ is measured based on the following equation.

$$|Pa_j| = \frac{M_p}{\Delta_j} \tag{5.3}$$

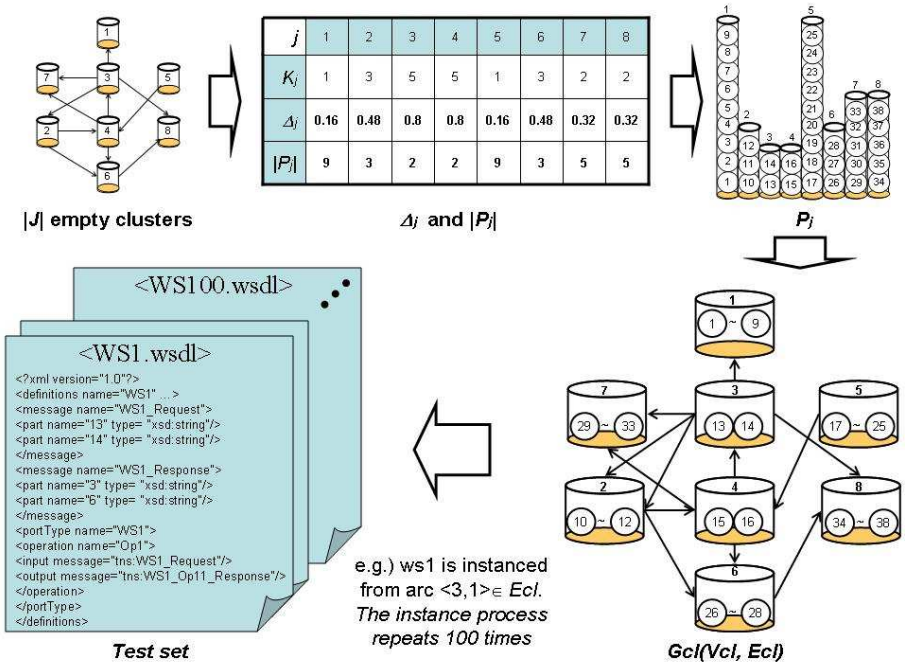
where Pa_j is the set of parameters contained in cluster j.

(4) For each j cluster, atomic parameters are generated up to $|Pa_j|$, with duplicated parameters forbidden (i.e., $\forall i, j \in V_{cl}$, $Pa_i \cap Pa_j = \emptyset$).

Once a complete parameter cluster network, $G_{cl}(V_{cl}, E_{cl})$ is built, WSBen repeats the following procedure until $|W|$ number of web services are generated:

1. A web-service template $\langle i, j \rangle$ is chosen at random from E_{cl}.

2. WSBen generates a WSDL file, in which each input parameter is selected from Cluster i with probability $\Delta_i \eta$, and each output parameter is selected from Cluster j with probability $\Delta_j \eta$.



j	1	2	3	4	5	6	7	8
K_j	1	3	5	5	1	3	2	2
Δ_j	0.16	0.48	0.8	0.8	0.16	0.48	0.32	0.32
$\|P_j\|$	9	3	2	2	9	3	5	5

Figure 5.2: Test set generation with ⟨ 8,Barabasi-Albert(8,2),0.8,1.5,100 ⟩

In Section 5.2, we illustrate each step described above through sample examples.

5.2 Illustration of WSBen

Figure 5.2 illustrates how WSBen builds G_{cl} and generates WSDL files based on the G_{cl}. Suppose that $xTS = \langle 8,\text{Barabasi-Albert}(8,2),0.8,1.5,100\rangle$ is given. Then, the generation steps are as follows:

1. WSBen generates a graph of Barabasi-Albert(8,2). The direction of each edge is determined at random. This means that WSBen does not have an assumption of whether incoming and outgoing edge degree of a parameter cluster are characterized by symmetric or asymmetric form. In practice, it is a more reasonable assumption that the incoming and outgoing edge degree of a parameter cluster is asymmetric. For example, a vast number of web services use "zip-code" as their input parameter, but web services with "zip-code" in their output parameter are sparse. Consequently, the incoming edge and outgoing edge degree of $i \in V_{cl}$ tend to be asymmetric. Nonetheless, the

current implementation of WSBen determines the direction of edges simply at random, as there has been no concrete research or evidence to discover this asymmetric property between the incoming and outgoing edge degrees of a parameter cluster.

2. Δ_j and $|Pa_j|$ are specified. For example, Cluster 5 has nine parameters as shown in Figure 5.2. That is, $|Pa_5| = 9$, as $\Delta_5 = \frac{k_j}{\max_{j \in V_{cl}} k_j} \times \eta = \frac{1}{5} \times 0.8 = 0.16$, resulting in $|Pa_5| = \frac{M_p}{\Delta_t} = \frac{1.5}{0.16} \simeq 9$.

3. Pa_j is specified. For example P_5={"17","18","19","20","21","22","23","24","25"} as shown in Figure 5.2 because $|P_5| = 9$ and for $\forall j \in J$, $P_5 \cap Pa_j = \emptyset$. Note that the parameter names are automatically generated, and thus do not contain any semantics.

4. Finally, G_{cl} is built and WSBen generates $|W|$ web services by:

   (a) randomly choosing $\langle i, j \rangle \in E_{cl}$.

   (b) selecting input parameters from cluster i with the co-occurrence probability of Δ_i, and output parameters from cluster j with the co-occurrence probability of Δ_j.

   For example, in Figure 5.2, `ws1` is instantiated from a web-service template $\langle 3, 1 \rangle \in E_{cl}$. Note that $\Delta_1 = 0.16$ and $\Delta_3 = 0.8$. $\Delta_1 = 0.16$ suggest that the occurrence probability of each parameter in Cluster 1 has 0.16. Due to the low probability, only "1" and "9" are selected from Cluster 1. Similarly, $\Delta_3 = 0.8$ means that the occurrence probability of each parameter in Cluster 3 has 0.8. Due to the high probability, all parameters in Cluster 3 that are "13" and "14" are selected. Note that each parameter in one cluster can map into either an input parameter or an output parameter. In the case that no parameter is generated, dummy parameters "S" and "T" are filled in the input and output parameters, respectively.

For experimental purposes in Chapter 7, we build three test set frameworks by specifying xTS as follows:

(1) $baTS = \langle$ 100,Barabasi-Albert(100,6),0.8,5,$|W|$ $\rangle$

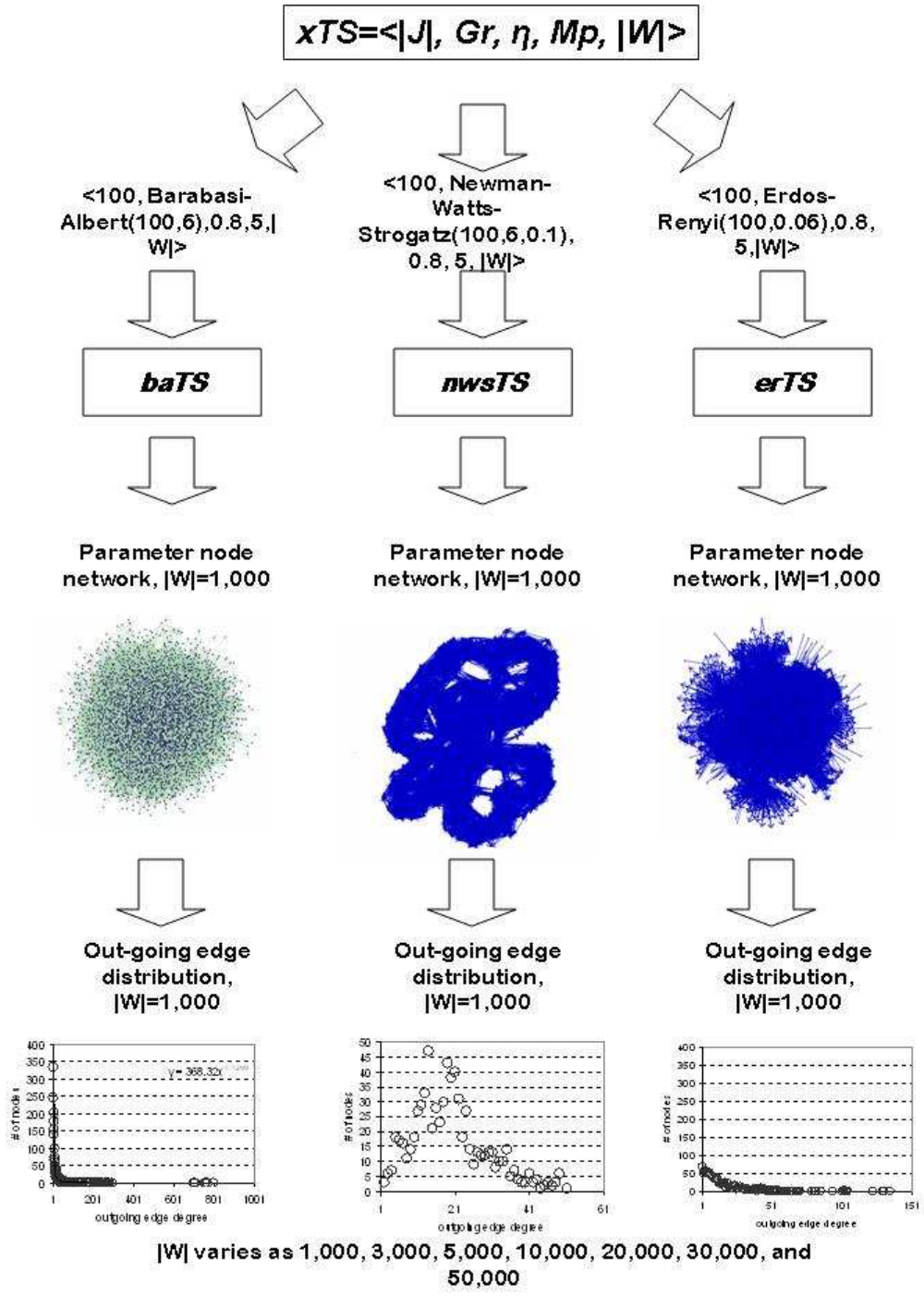


Figure 5.3: Overview of $baTS$, $nwsTS$, and $erTS$

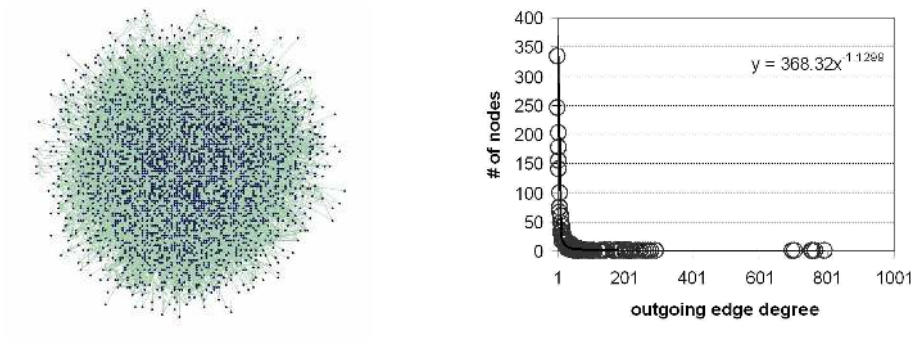


Figure 5.4: G_p of baTS at $|W| = 1,000$. (left) G_p. (right) outgoing edge degree distribution

(2) $nwsTS = \langle$100,Newman-Watts-Strogatz(100,6,0.1),0.8,5,$|W|\rangle$

(3) $erTS = \langle$100,Erdos-Renyi(100,0.06),0.8,5,$|W|\rangle$

Note that if the first four tuples are grounded, G_{cl} can be generated. For each G_{cl} of $baTS$, $nwsTS$, and $erTS$, seven different sized test sets are generated by varying $|W|$ as 1,000, 3,000, 5,000, 10,000, 20,000, 30,000, and 50,000, respectively. Consequently, 21 test sets are prepared (three frameworks $\times$ seven different test sizes). Each G_{cl} of the three test set frameworks has a different $|P|$. For example, $baTS$ has 4,231 while $nwsTS$ and $erTS$ have 751 and 1,392, respectively. Figure 5.3 shows the G_p and its outgoing edge degree distribution for each of $baTS$, $nwsTS$, and $erTS$, when $|W| = 1,000$. Each of the 21 test sets has five test requests. The test request r is constructed such that r^o is farthest away from r^i in a parameter space. The procedure of generating requests will be discussed in more detail in Chapter 7.

5.2.1 Characteristics of $baTS$

We can regard the scale-free network properties of $baTS$ web service networks. Figure 5.4, 5.5 and 5.6 show G_p, G_{op} and G_{op}^f of $baTS$ at $|W|$=1,000, along with their outgoing edge degree distributions. In the outgoing edge degree distribution of each web-service network, the x-axis represents the number of outgoing edges and the y-axis represents the number of the node with the same outgoing edges. In order to check the scale-free property, we can use the power function $P_w(v)$, where

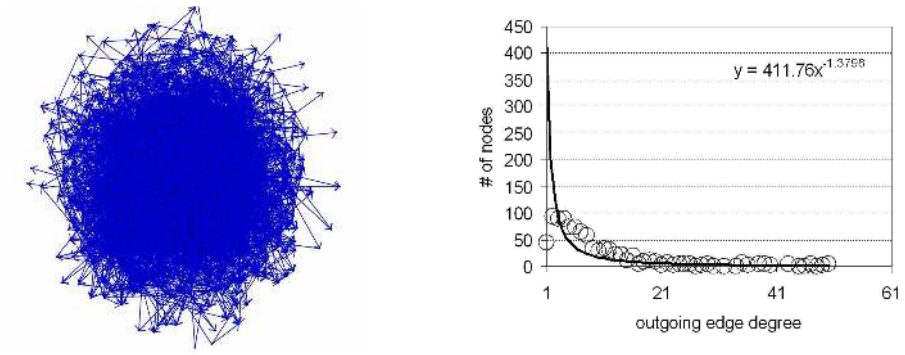


Figure 5.5: G_{op} of baTS at $|W| = 1,000$. (left) G_{op}. (right) outgoing edge degree distribution

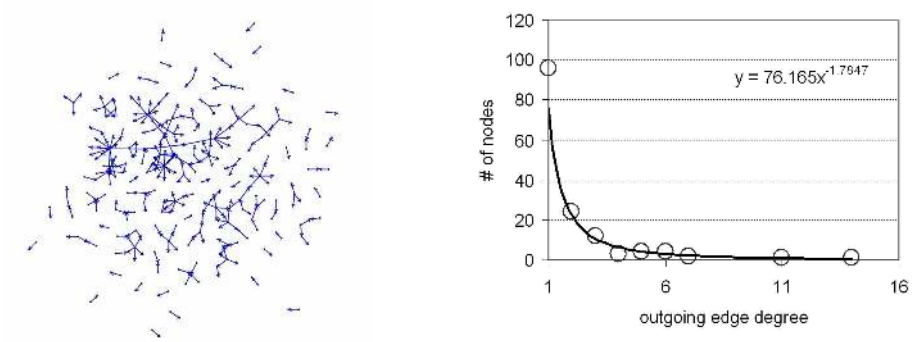


Figure 5.6: G^f_{op} of baTS at $|W| = 1,000$. (left) G^f_{op}. (right) outgoing edge degree distribution

$P_w(v) \propto v^{(-\gamma)}$. We apply $P_w(v)$ to each outgoing edge degree distribution and check γ, the exponent value of $P_w(v)$. In this case, $P_w(v)$ represents the number of nodes that have v number of neighbor nodes. Note that if γ is greater than two and there is no hump, then it is evident that the distribution follows the power law and has the scale-free property. The γ values of G_p and G^f_{op} are 1.1299 and 1.7847, respectively. The values do not suffice the requirement to assert that the distributions follow the power law [32]; the distributions are skewed highly enough to be regarded as Zipf distributions[4]. Regarding G_{op}, the outgoing edge degree distribution of G_{op} clearly shows a hump at $x = 2$. Due to the hump, its γ value has no meaning as far as the scale-free property is concerned.

When it comes to small-world properties, Table 5.2 shows the average path

[4]See Chapter 4 for the definition of the Zipf's law and the Zipf distribution.

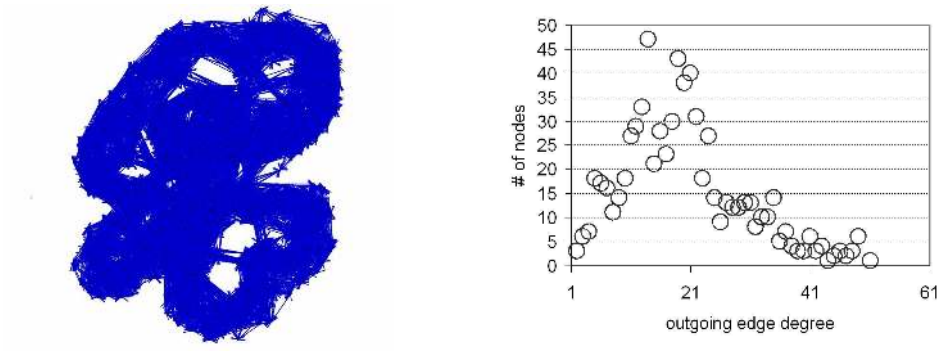


Figure 5.7: G_p of nwsTS at $|W| = 1,000$. (left) G_p. (right) outgoing edge degree distribution

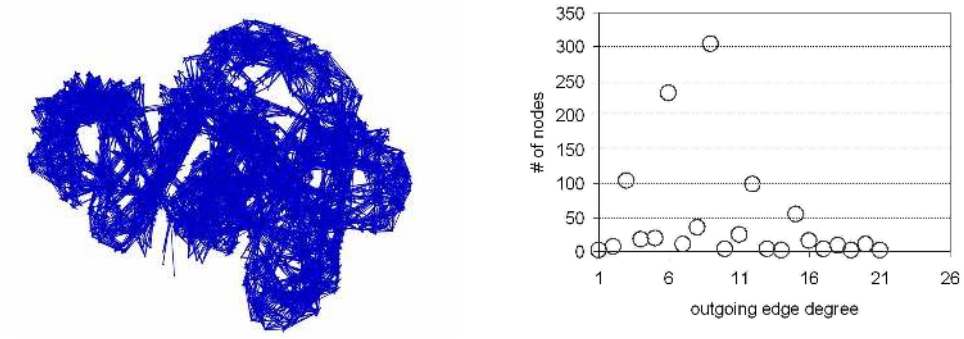


Figure 5.8: G_{op} of nwsTS at $|W| = 1,000$. (left) G_{op}. (right) outgoing edge degree distribution

length L and clustering coefficient C for three giant connected components extracted from each of G_p, G_{op}, and G^f_{op}. These are compared to random graphs with the same number of nodes and the average number of edges per node. Note that we treat all edges of each network of G_p, G_{op}, and G^f_{op} as undirected and un-weighted as we did in Chapter 4. The information about original web-service networks and their giant components are summarized in Table 5.1 and Table 5.3, respectively. The results show that G_p and G_{op} are close to random networks because $L \succeq L_{random}$ and $C \succeq C_{random}$. On the contrary, G^f_{op} becomes a tree. That is, $C = 0$.

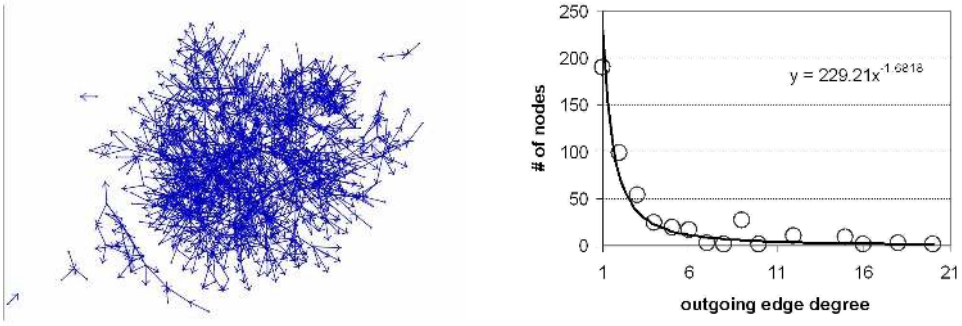


Figure 5.9: G^f_{op} of nwsTS at $|W| = 1,000$. (left) G^f_{op}. (right) outgoing edge degree distribution

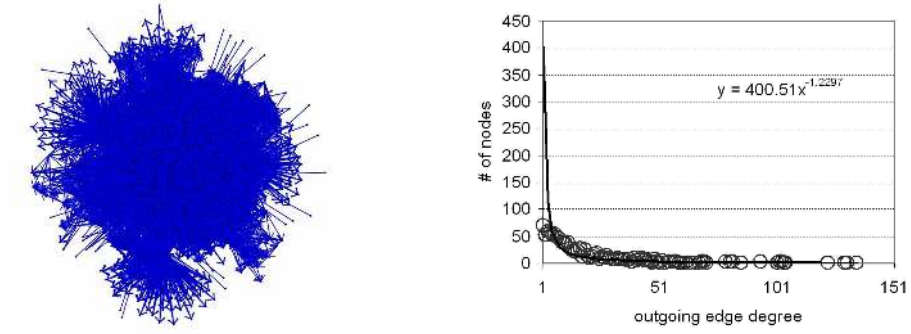


Figure 5.10: G_p of erTS at $|W| = 1,000$. (left) G_p. (right) outgoing edge degree distribution

Table 5.1: Features of web-service networks in $baTS$

Features	G_p	G_{op}	G^f_{op}
# of nodes	3,553	992	319
# of arcs	33,969	8,141	275
Network Diameter	10	9	4

Table 5.2: Small-world properties of web-service networks in $baTS$

Networks	L_{actual}	L_{random}	C_{actual}	C_{random}
G_p	3.1487	3.0475	0.0351	0.0052
G_{op}	3.3449	2.7508	0.0041	0.0154
G^f_{op}	4.4588	3.3549	0	0.0263

5.2.2 Characteristics of $nwsTS$

In succession to the previous observation on $baTS$, we can continue to highlight the characteristics of $nwsTS$ in terms of scale-free and small-world network proper-

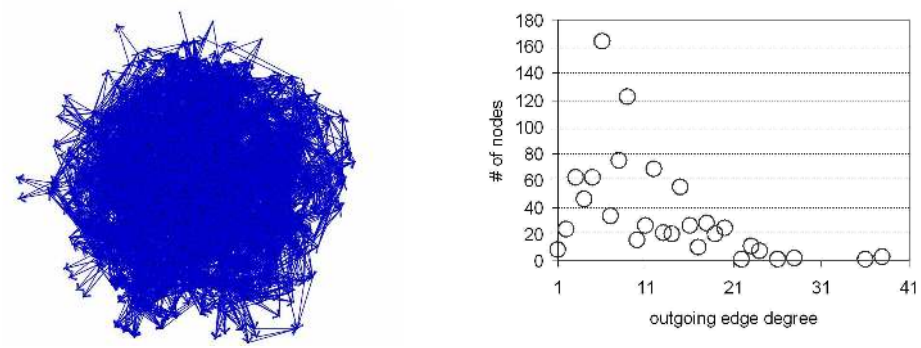


Figure 5.11: G_{op} of erTS at $|W| = 1,000$. (left) G_{op}. (right) outgoing edge degree distribution

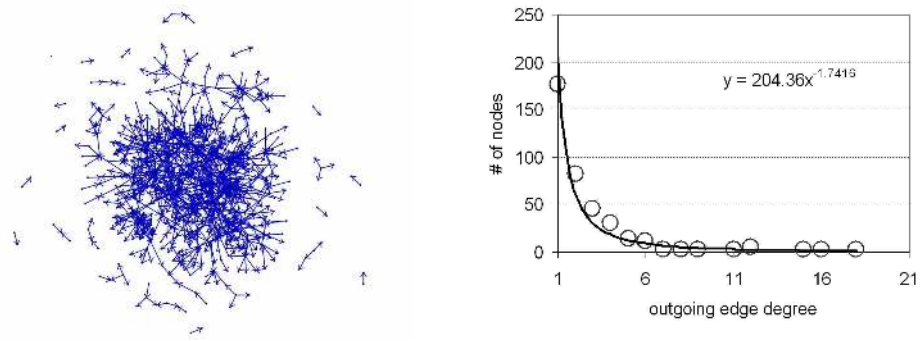


Figure 5.12: G^f_{op} of erTS at $|W| = 1,000$. (left) G^f_{op}. (right) outgoing edge degree distribution

Table 5.3: Summary of giant components in $baTS$ web-service networks

Networks	# of nodes(A)	# of nodes in giant component(B)	$\frac{B}{A} \times 100(\%)$	average # of edges per node in giant component
G_p	3,553	3,553	100(%)	19.12
G_{op}	992	992	100(%)	16.41
G^f_{op}	319	39	12.22(%)	2.25

ties. First, we can check the scale-free properties of $nwsTS$ web-service networks. Figure 5.7, 5.8 and 5.9 show G_p, G_{op}, and G^f_{op} of $nwsTS$ at $|W|$=1,000, along with their outgoing edge distributions. The outgoing edge degree distribution of G^f_{op} with the γ value of G^f_{op}, 1.6818 is very skewed. In other words, there are many low

Table 5.4: Features of web-service networks in $nwsTS$

Features	G_p	G_{op}	G^f_{op}
# of nodes	751	997	823
# of arcs	14,392	8,102	1,474
Network Diameter	18	19	25

Table 5.5: Small-world properties of web-service networks in $nwsTS$

Networks	L_{actual}	L_{random}	C_{actual}	C_{random}
G_p	3.6951	2.0818	0.3661	0.0511
G_{op}	5.4085	2.7544	0.0348	0.0159
G^f_{op}	7.9318	5.3216	0.0031	0.002

x-values (outgoing edge degree) and few very high x-values. It implies that there are a few hub operations, or web service, with large degree. It is interesting to view that G_p and G_{op} have bell shape distributions. This distribution is expected because when we form the parameter cluster network of $nwsTS$, the probability to add new link to be 0.1 at which we previously demonstrated that small-world properties arise, as shown in Figure 4.4. Due to small-world properties, $nwsTS$ partially holds the property of a regular network, which has the constant number k of edges per node. Consequently, the distributions show that the more frequent values are near the middle of their distributions which is formed due to the k value, and the frequency tapers off gradually near the high and low extremes of the range. Moreover, since the initial regular network of the Newman-Watts-Strogatz model has the circular form, G_p and G_{op} retains the original circular form partially, as shown in Figure 5.7 and Figure 5.7.

Regarding small-world properties, Table 5.5 shows the average path length L and clustering coefficient C for three giant connected components extracted from each of G_p, G_{op}, and G^f_{op}, compared to corresponding random graphs. The information about original web-service networks and their giant components are summarized in Table 5.4 and 5.6, respectively. The results show that G_p has small-world properties: $L \succeq L_{random}$ and $C \gg C_{random}$. G_{op} and G^f_{op} are close to a random network because $L \succeq L_{random}$ and $C \succeq C_{random}$.

Table 5.6: Summary of giant components in $nwsTS$ web-service networks

Networks	# of nodes(A)	# of nodes in giant component(B)	$\frac{B}{A} \times 100(\%)$	average # of edges per node in giant component
G_p	751	751	100(%)	38.32
G_{op}	997	997	100(%)	16.25
G_{op}^f	823	782	95.01(%)	3.65

5.2.3 Characteristics of $erTS$

Following the previous observation of $nwsTS$, we can continue to deal with web-service networks of $erTS$ to examine their scale-free and small-world properties. Figure 5.10, 5.11, and 5.12 show G_p, G_{op}, and G_{op}^f of $erTS$ at $|W|$=1,000, along with their outgoing edge degree distributions. The outgoing edge degree distribution of G_{op}^f with γ=1.7418 can be regarded as highly skewed toward the low extreme of the range. This finding implies that a few hub operations (web services) exist with large degree, but there are many operations with small edges.

It is interesting to view that G_p and G_{op} have different shapes in their distributions. As a whole, G_p has a uniformly shaped distribution while G_{op} has a Zipf-like distribution. Note that WSBen assumes that the connectivity of a parameter cluster is in inverse proportion to the number of parameters in the cluster node. Therefore, if a cluster j in G_{cl} has a small connection with other clusters, then j has large $|Pa_j|$; i.e., a large number of parameters are contained in j.

The random network model generates G_{cl} such that the connection between clusters are linked at random allowing the connectivity distribution to follow the Law of Large Numbers statistically. Note that we set the probability of choosing each of the possible edges between clusters in $erTS$ to be 0.06 but the value of 0.06 is not too large to cause the Law of Large Numbers because the average edges per cluster becomes just six(= $|J| \times 0.06$, where $|J|$=100). Therefore, G_p has the uniformly shaped distribution. Regarding G_{op}, the outgoing edge degree distribution of G_{op} is affected by the size of the test set $|W|$. We pick the web-service templates (the links between clusters) as many as 1,000 ($|W|$=1,000), and the value of 1,000 is enough to cause the Law of Large Numbers so that G_{op} can

Table 5.7: Features of web-service networks in $erTS$

Features	G_p	G_{op}	G_{op}^f
# of nodes	1,332	997	669
# of arcs	17,006	8,946	983
Network Diameter	13	12	8

Table 5.8: Small-world properties of giant components in $erTS$ web service networks

Networks	L_{actual}	L_{random}	C_{actual}	C_{random}
G_p	3.2703	2.5776	0.0228	0.0189
G_{op}	3.8451	2.7051	0.0044	0.017
G_{op}^f	4.4588	3.3549	0	0.0263

Table 5.9: Summary of giant components in $erTS$ web-service networks

Networks	# of nodes(A)	# of nodes in giant component(B)	$\frac{B}{A} \times 100(\%)$	average # of edges per node in giant component
G_p	1,332	1,332	100(%)	25.53
G_{op}	997	8,946	100(%)	17.94
G_{op}^f	669	598	89.38(%)	3.13

have the bell-shaped distribution.

Regarding small-world properties, Table 5.8 shows the average path length L and clustering coefficient C for three giant connected components extracted from each of G_p, G_{op}, and G_{op}^f, compared to corresponding random graphs. The information about original web-service networks and their giant components are summarized in Table 5.7 and 5.9, respectively. As a whole, G_p and G_{op} have the random network property, while G_{op}^f becomes a tree and the clustering coefficient $C = 0$.

5.2.4 Estimating the Size of Giant Component

One can study the properties and behaviors of a network by synthetically generating the network and estimating its various properties. There exists a large variety of network models roughly categorized as “random”, “small-world” and

"scale-free" types and these have been shown to model many real-world networks sufficiently [2]. In this section, as a demonstration, we attempt to estimate the size of giant component in semantic web services networks using random graph theory. Often it is believed to be important to have a large and dense giant component in a service network. Otherwise, isolated services nodes will never have a chance to provide any services to clients.

A random graph is simple to define. One takes N nodes and places edges between each pair with a probability p. This simplest model [39] is certainly the best-studied but not a proper model for real-world networks. It generates a poisson degree distribution for large N. However, any real-world network can be represented by a random network created in a semi-random fashion by taking the degree distribution of the network into account. That is, we are given a degree distribution p_k which is the probability that a randomly chosen node has degree k. We can make a model network with the same degree distribution as follows: we take a number of nodes and create k ends of edges for these nodes, where k is a random number drawn independently from the distribution p_k. Then we can randomly select in pairs from these ends of edges to create edges between them. The procedure will create a random graph with the desired degree distribution [78].

Many properties of such a random network model, including the size of a giant component, are shown to be exactly solvable in [74, 75, 78] in the limit of large network size. Here we use the theoretical framework derived in [78] in order to estimate the giant component size in sample networks created by WSBen by using generating functions [122]. The idea is instead of dealing with the degree distribution directly, a generating function $G_0(x)$ that encapsulates all the information in the degree distribution p_k is used as follows:

$$G_0(x) = \sum_{k=0}^{\infty} p_k x^k \tag{5.4}$$

where k presents degree and p_k is the desired degree distribution of the network. The probability p_k can be obtained by the kth derivative of G_0 using:

$$p_k = \frac{1}{k!} \frac{d^k G_0}{dx^k}\bigg|_{x=0} \tag{5.5}$$

So all information contained in the discrete probability function p_k can be represented by one G_0 generating function. This form is easier to work on rather than working with the complexities of the actual distribution function. We can, for instance, represent average degree, z, for any degree distribution using this generating function as:

$$z = \sum_k k p_k = G_0'(1) \tag{5.6}$$

One important point towards the analysis of component sizes, we need to be able to represent degree distribution of node we reach by following a randomly chosen edge. It is not the same as p_k since there are k edges that arrive at a node with degree k. Hence, it is proportional to kp_k, and can be generated by the following normalized function:

$$\frac{\sum_k k p_k x^k}{\sum_k k p_k} = x \frac{G_0'(x)}{G_0'(1)} \tag{5.7}$$

If we start from a randomly chosen node and follow its edges to reach its neighbors, we need to subtract the edge connecting the node to the neighbor in order to find the degree distribution of the remaining outgoing edges of the neighbors of that node. Thus we can define the generating function for the degree distribution of a randomly chosen node's neighbors $G_1(x)$ as:

$$G_1(x) = \frac{G_0'(x)}{G_0'(1)} = \frac{1}{z}\, G_0'(x) \tag{5.8}$$

The average size of components to which a randomly chosen node belongs when there is no giant component formed in a network is derived in [78] as:

$$\langle s \rangle = 1 + \frac{G_0'(1)}{1 - G_1'(1)} = 1 + \frac{z_1^2}{z_1 - z_2} \tag{5.9}$$

where $z_1 = z$ is the average number of neighbors of a node and z_2 is the average number of second neighbors. The expression diverges when $G_1'(1) = 1$ that is the point marking a phase transition at which a giant component first appears. By

rewriting the conditions, we can say a giant component exists in a network if

$$\sum_k k(k-2)p_k \geq 0 \tag{5.10}$$

The size of giant component, if there is one, can be calculated from the following simple heuristic argument. Let u be the probability that a node chosen uniformly random from the network is not in the giant component. In other words, this value is the fraction of all nodes outside the giant component. Then this probability is equal to the probability that none of the node's neighbors belong to the giant component which is just u^k if the node has degree k. If we average this over the probability distribution [78], we have the following function for u:

$$u = G_1(u) \tag{5.11}$$

then for the smallest non-negative real solution of u, the following Equation gives the size of the giant component, S:

$$S = 1 - G_0(u) \tag{5.12}$$

In order to see if this theoretical framework works, we generated 10 g_{op}^f with increasing network sizes for each of following cases:

1. Random: $\langle$50,Erdos-Renyi(100,0.06),0.8,5,$|W|\rangle$
2. Scale-free: $\langle$50,Barabasi-Albert(100,6),0.8,5,$|W|$ $\rangle$
3. NWS: $\langle$50,Newman-Watts-Strogatz(100,6,0.1),0.8,5,$|W|\rangle$

Figure 5.13 shows how g_{op}^f changes by increasing $|W|$ by 100 for each of random, scale-free, and NWS cases.

For each of these networks, we measured the size of the giant component and checked the phase transition using the above random network model. Expectedly, the degree distribution of each network met the phase transition threshold given by Equation 5.10. Thus we also calculated the size of the giant component theoretically according to Equations 5.11 and 5.12 and compared with measured sizes for

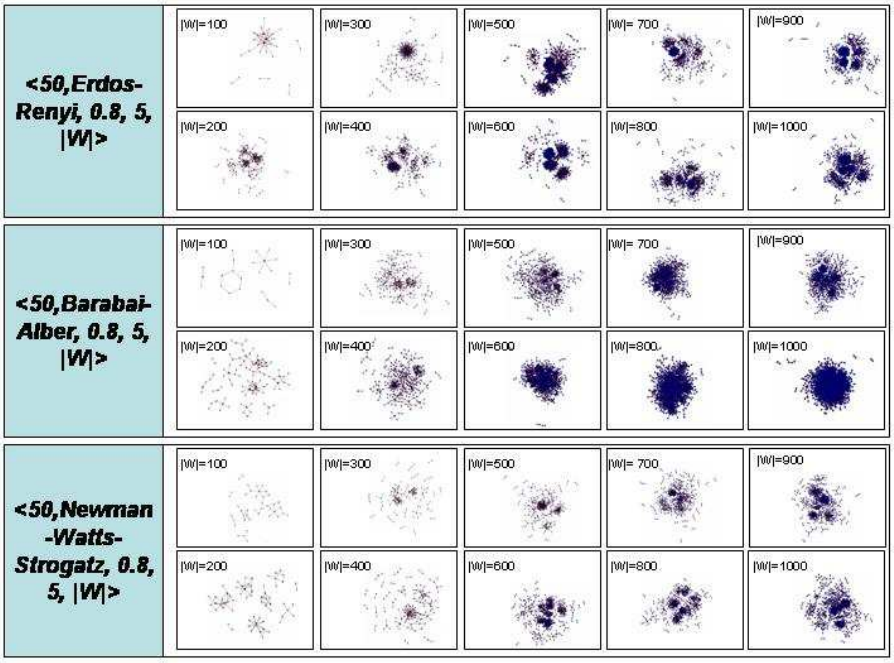


Figure 5.13: How g_{op}^{f} changes by increasing $|W|$

each generated network. The comparisons are shown Figure 5.14 as a sub-figure for each network topology type.

For g_{op}^{f} based on the random parameter cluster network in 5.14-A, the theoretical value of the giant component size is very close to the measured one for each synthetic network. This implies that even a simple random model may be very helpful to estimate the inter-operable portion of such networks with random topology without even analyzing the available network beyond its degree distribution. This is expected since the model is already aimed for such networks. However, Figure 5.14-B shows that it is not a good scale-free model. There is a considerable gap between theory and real value for many of the synthetic networks in this type.

A prominent phenomenon called “preferential attachment” [2] is the main mechanisms in scale-free network type for selecting a node for attaching an edge from an origin node. This causes many nodes with small degrees, and a few nodes, named ‘hub’ nodes, with large degree which are usually connected to many others nodes through the network. Thus this model generates a power-law degree distribution which is quite a deviation from a poisson distribution generated by a

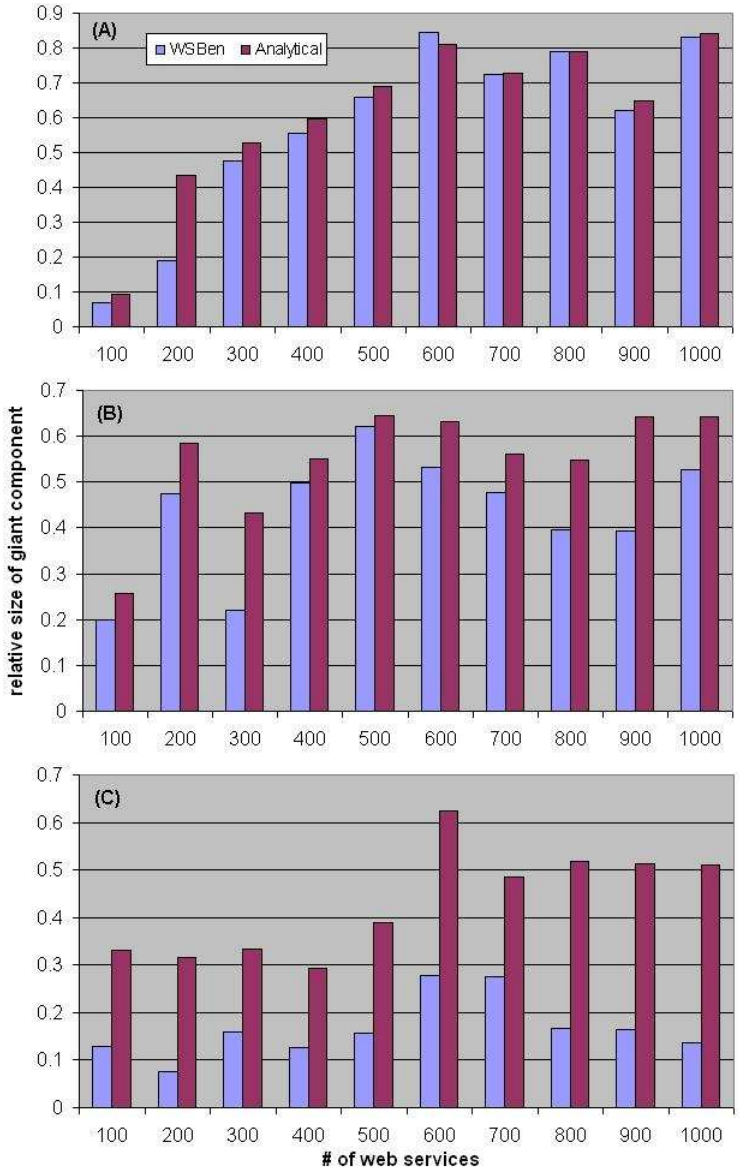


Figure 5.14: Comparison of actual and estimated size of giant components. (A: Random , B: Scale-free, C: NWS)

random model. However, aside from node selection for edges, the model is random in all other ways. This could be the reason for the agreement for a few network samples in this type. Nevertheless, the lack of enough randomness in the model creates a gap in the theoretical and actual measurements. For network samples with larger number of web services, the emergence of hub nodes should be more apparent. This appearance makes the topology of these networks fit better into

a power-law degree distribution, implying that a random model fails to estimate the size of the giant component sufficiently. If these sample networks were also effectively random all over then one would expect the model to agree perfectly with actual measurements.

The deviation between theory and actual network results becomes more dramatic for the NWS (small world) type shown in Figure 5.14-C. Unlike the previous two types, a NWS network starts with a regular topology and randomness is introduced into this regularity to a certain degree defined by the probability p which is 0.1 in our case. Unless $p = 1$, this initial topology remains unaffected and still regular in parts of the network.

The results show that a random network model might be a good generative model for such web-service networks if these networks are entirely random, which is also the basic assumption in the model [78]. Also, the results can be calculated exactly in the limit of large network size, unlike the networks used in this experimentation. Another important point is that for a given probability distribution, there is a set of random networks which fit the given random distribution. The theoretical calculations here should be considered as an average behavior over all such networks.

In this chapter, we introduced WSBen by which we generated test sets in such a way that their underlying parameter cluster networks and their test set sizes varied. We also demonstrate the application of WSBen by illustrating a use case in the network theory community. In the following chapter, we will present WSPR which is the AI planning-based heuristic algorithm. We will test WSPR as well as other prominent AI planning algorithms using the test sets generated by WSBen in Chapter 7.

Chapter 6

WSPR: Web Service PlanneR Algorithm

In this chapter, we present the AI planning-based two step heuristic algorithm titled WSPR. To highlight the benefits of a two-step guided search, we compare WSPR with two A*-variant algorithms that use heuristics based on a best-first forward search from the initial state to the goal. The two A*-variant algorithms are only used as a baseline to compare WSPR, and we do not explore the two algorithms further in subsequent chapters.

6.1 WSPR algorithm

In Chapter 2, we show that the size of the state space is exponential to the size of the parameter set, and formulate the state model for the WSC problem as $\Psi = \langle S, s_0, S_G, \Omega(.), f, c \rangle$, where:

(1) The state, $s \in S$ is a collection of parameters in P,

(2) The initial state $s_0 \in S$ is such that $s_0 = r^i$,

(3) The goal states $s \in S_G$ are such that $r^o \subseteq s$,

(4) $\Omega(s)$ is the set of web services $w \in W$ such that $w^i \in s$. That is, w can be invoked or applicable in the state s,

(5) The transition function $f(w, s) = s'$ that maps a state s into a state s' such that $s' = s \cup w^o$ for $w \in \Omega(s)$, and

(6) $c(w)$ is the invocation cost of w.

To address this intractable WSC problem, we suggest a polynomial-time approximation algorithm, *WSPR*[1]. When a request r is given, WSPR activates its two-step search. First, it computes the cost of achieving individual parameters starting from r^i by conducting a forward search; second, it approximates the optimal sequence of web services that connects r^i to r^o by conducting a regression search, leveraging on the results obtained from the first step as guidance.

This two-step based approach is essentially in accordance with Graphplan [19]. However, our method is different in the sense that we use novel heuristics to minimize the number of web services in a solution. In contrast, Graphplan and other AI planners originating from Graphplan typically aim at minimizing the number of time steps, but not necessarily the number of actions (web services). Note that there can be a crossover between STRIPS and the state models. For example, in the forward search, we have to consider that $s_0 = r^i$ and $s_g \supseteq r^0$ where $s_g \subseteq S_G$, and in the regression search, we have to consider that $s_g = r^o$ where $s_g \subseteq S_G$ and $s_0 \subseteq r^i$. s_0, s_g, and S_G are terms for the state model, while r^i and r^o are used in the STRIPS model. Since this crossover can cause confusion, we will use r^o and r^i rather than s_0 and S_G in the algorithm. However, we will use $s \in S$ to trace the current information state in the algorithm. Therefore, it must be notified that state s in the progression space has a different meaning than that in the regression space. In the progression space, states can be thought as sets of preconditions, but in the regression space, states can be thought as sets of effects. The differences are described in detail below.

(Step1) Forward Search: In the first stage, WSPR obtains $g_{r^i}(p)$ - the cost of achieving $p \in P$ from a state r^i. This cost can be characterized by the solution of a recursive equation as follows:

$$g_{r^i}(p) = \min_{w \in Ow(p)} [c(w) + \max_{p' \in w^i} g_{r^i}(p')] \tag{6.1}$$

[1]WSPR is pronounced like "whisper"

where $c(w)$ is an invocation cost of a web service and is assumed to be 1[2]. $Ow(p)$ is a set of web services: $Ow(p) = \{w \in W | p \in w^o\}$. At first, $g_{r^i}(p)$ is initialized to 0 if $p \in r^i$, and to ∞ otherwise. Then, the current information state s is set to r^i (Line 1 in Algorithm 1). Every time for $\forall w \in \Omega(s)$, each parameter $p \in w^o$ is added to s, and $g_{r^i}(p)$ is updated until for $\forall p \in r^o$, $g_{r^i}(p)$ are obtained (Lines 2-6 in Algorithm 1). If $\Omega(s)$ does not increase any more, there remains no additional search space meaning that no solution exists. We name a web service w as a predecessor web service of $p \in P$ if w is the first web service to generate p. We denote $PD_{ws}(p)$ to be an inverted index [30] that contains the set of predecessor web services of p. In this thesis, we assume that the invocation cost of web services is non-negative. However, it is possible to have a negative web-service invocation cost if various Quality of Service (QoS) are considered (e.g., cost, quality, security). In that case, rather than Equation (6.1), label-correcting algorithms such as the Bellman-Ford algorithm [30] must be used.

```
   Input  : r^i and r^o
   Output: PD_ws
1  s = (r^i); C = ∅; d = 1;
2  while ¬(s ⊇ r^o) do
3      δ={w|w ∈ Ω(s), w ∉ C};
4      for p in w^o (w ∈ δ) do
5          if g_{r^i}(p) = ∞ then
6              g_{r^i}(p) = d; PD_ws = w; s = s ∪ {p};
7      C = C ∪ δ; d++ ;
```

Algorithm 1: Forward search algorithm of WSPR

(Step2) Regression Search: In the second stage, WSPR approximates the optimal sequence of web services that connects r^i to r^o by conducting the regression search, as directed by $g_{r^i}(p)$ and $PD_{ws}(p)$, which are obtained from the first

[2]We assumed that for all web services w, the invoking cost $c(w)$ is 1 identical. In a real system, however, each web service may have distinct $c(w)$. $c(w)$ could be determined based on real market surveys, or using a pricing model specific to web services, suggesting that $c(w) = 1$ can be relaxed such that $c(w) > 0$. This relaxation has an impact on WSPR, because current WSPR's heuristic measures the contribution of a web service based on how many of its output parameters overlap with a sub-goal. The detailed modification of WSPR required to address this relaxation is described in Chapter 9.2

step. In this thesis, we propose a greedy algorithm-based backward search. The backward search is an old idea in planning that is also known as *regression search*. In regression search, state s can be thought as a set of effects and we can specify a sub-goal from state s. This algorithm denotes its sub-goal by *subGoal* and sets *subGoal* to (r^o) in the beginning (Line 1 in Algorithm 2). We can denote *wSpace* to be a set of web services $w \in W$, such that $w_i \in PD_{ws}(p)$, where $p \in subGoal$. Then, WSPR selects a web service from *wSpace* by considering their heuristics at each backward step (Lines 3-6 in Algorithm 2). This backward selection procedure is repeated until $subGoal \subseteq r^i$ (Line 2 in Algorithm 2). The heuristics used for selecting a web service and its underlying hypothesis is as follows:

Hypothesis-1: *Choosing a web service with a greater contribution to match subGoal earlier in the regression search helps reach the initial state faster.*

$$h_{sg}(w) = |w^o \cap sugGoal| \tag{6.2}$$

$h_{sg}(w)$ implies that WSPR favors a web service with a bigger contribution to match the sub-goal. However, $h_{sg}(w)$ has another important interpretation. A web service w with bigger $h_{sg}(w)$ has a higher probability to match the sub-goal fully, leading to preventing following search space to expand significantly. In other words, our heuristics attempts to avoid a partial matching case, or reduce the size of partial matching web services as much as possible.

In the case that multiple web services with the same $h_{sg}(w)$ value are in the OPEN list [30], we choose a web service with the largest edge degree in the web-service node network, to break the tie situation. The use of this tie-break rule has an impact to increase the chance of finding better web services by fertilizing the succeeding search space, because the web service with the larger edge degree is likely to be connected to more web services.

In Chapter 7, we assess the performance difference provided by the WSPR heuristics. To this end, we compare the effectiveness and efficiency of WSPR with those of WSPR without the heuristics.

Input : r^i, r^o, and PD_{ws}
Output: $w_1 \Rightarrow \cdots \Rightarrow w_n$
1 $subGoal = r^o$;
2 **while** $\neg(subGoal = \emptyset)$ **do**
$wSpace = \bigcup_{p \in subGoal} PD_{ws}(p)$;
3
$\chi = \arg \max_{w \in wSpace} h_{sg}(w)$;
4
5 $soln = soln \cup \chi$;
6 $subGoal = [subGoal \setminus (\chi^o \cup r^i)] \cup \chi^i$;
7 $s = r^i$;
8 **while** $\neg(soln = \emptyset)$ **do**
9 **if** $w \in \Omega(s)$ *and* $w \in soln$ **then**
10 Print w, "$\Rightarrow$";
11 $s = s \cup w^o$;
12 $soln = soln \setminus w$;

Algorithm 2: Regression search algorithm of WSPR

6.1.1 Analysis of the WSPR algorithm

In this section, we analyze the WSPR's computational complexity and prove the WSPR's correctness as well as an illustration of WSPR using the motivating example of Chapter 1.

The forward search procedure has the polynomial computation time $O(|W|^2|P|)$. First, the length of a sequence of web services to satisfy a request is limited by $|W|$. Therefore, there are at most $|W|$ iterations. Second, at each iteration of the forward search, the maximum $|W|$ web services and $|P|$ parameters are examined. Consequently, the computational complexity of the forward search procedure costs $O(|W|^2|P|)$. On the contrary, the regression searching procedure has the polynomial computation time $O(|W|^2 log|W|)$. First, the regression search procedure has at most $|W|$ iterations, and at each iteration, the maximum $O(|W|log|W|)$ time is required to conduct the sorting task to select a web service w with the largest $h_{sg}(w)$. The time taken to print a solution can be ignored (Line 8-12 in Algorithm 2). In general, $|P| \gg log|W|$, so that $O(|W|^2|P|) \gg O(|W|^2 log|W|)$. In other words, the forward search takes much longer computational time than that of the regression search. As a result, the performance of WSPR is dominated by the

forward search procedure. We verify this insight experimentally in Appendix B. In this remark, we find that there are three significant dimensions to determine the performance of WSPR: (1) The length of a sequence web service in a solution; (2) $|W|$ of the web service size; and (3) $|P|$ of the parameter set size.

With r_2 of the Atherton motivating example in Chapter 1, we can illustrate the forward search algorithm as follows:

- In the beginning, $s = r_2^i$. Since $\Omega(s) = \{\texttt{findHotel}\}$, we can invoke `findHotel`. We can update $g_{r^i}(p)$, $PD_{ws}(p)$, and s as follows:

  (1) g_{r^i}(“hotelAddress”)=1 and g_{r^i}(“hotelZipcode”)=1,

  (2) PD_{ws}(“hotelAddress”) = `findHotel`,

  (3) PD_{ws}(“hotelZipcode”) = `findHotel`,

  (4) $s = s \cup \{$“hotelAddress”,“hotelZipcod”$\}$.

- With an updated s, we can obtain $\Omega(s)= \{$`findRestaurant`, `guideRestaurant`$\}$. By invoking `findRestaurant` and `guideRestaurant`, we can maintain $g_{r^i}(p)$, $PD_{ws}(p)$, and s as follows:

  (1) g_{r^i}(“restaurantRate”) = 2,

  (2) g_{r^i}(“restaurantAddress”) = 2,

  (3) g_{r^i}(“restaurantName”) = 2,

  (4) g_{r^i}(“restaurantPhonenumber”) = 2,

  (5) PD_{ws}(“restaurantRate”) = {guideRestaurant},

  (6) PD_{ws}(“restaurantAddress”)= {guideRestaurant,findRestaurant},

  (7) PD_{ws}(“restaurantName”) = {findRestaurant},

  (8) PD_{ws}(“restaurantPhonenumber”) = {findRestaurant},

  (9) $s = s \cup \{$“restaurantRate”, “restaurantAddress”, “restaurantName”, “restaurantPhonenumber”$\}$.

- With an updated s, we can obtain $\Omega(s)=\{$`findDirection`$\}$. Also, we can maintain $g_{r^i}(p)$, $PD_{ws}(p)$, and s as follows:

(1) g_{r^i}(“mapHotelRestaurant”)=3,

(2) g_{r^i}(“directionHotelRestaurant”)=3,

(3) PD_{ws}(“mapHotelRestaurant”)={`findDirection`},

(4) PD_{ws}(“directionHotelRestaurant”)={`findDirection`},

(5) $s = s \cup$ {“mapHotelRestaurant”, “directionHotelRestaurant”}.

- We can stop the procedure because $s \supseteq r_2^o$.

We illustrate the regression planning algorithm, continued from the end of the forward search procedure above.

- In the beginning, $s = r_2^o$ and $subGoal$=s.
- Since PD_{ws}(“mapHotelRestaurant”) and PD_{ws}(“directionHotelRestaurant”) has the same value as {`findDirection`} , we immediately get χ = `findDirection`. In this case, we do not have to measure the heuristic value because only one web service is concerned. Then, we can update states as follows:

  (1) s=$s \cup$ {“restaurantAddress”, “hotelAddress”},

  (2) $subGoal$={“restaurantAddress”, “hotelAddress”},

  (3) $soln$={`findDirection`}.

- PD_{ws}(“restaurantAddress”) = { `guideRestaurant`, `findRestaurant`} and PD_{ws}(“hotelAddress”)={`findHotel`} and $wSpace$={ `findHotel`, `findRestaurant`, `guideRestaurant`}. Since each web service in $wSpace$ has $h_{sg}(w)$=1 uniformly, we can randomly select χ={`findHotel`}, and then we can update states as follows:

  (1) s=$s \cup$ {“hotelAddress”, “hotelZipcode”},

  (2) $subGoal$={“restaurantAddress”},

  (3) $soln$={ `findHotel`, `findDirection`}.

- PD_{ws}(“restaurantAddress”)={`guideRestaurant`, `findRestaurant`} and $wSpace$={`findRestaurant`, `guideRestaurant`}. Since each web service in $wSpace$ has $h_{sg}(w)$=1 uniformly, we can randomly select χ={ `findRestaurant`}, and then we can update states as follows:

(1) s=s∪{"hotelAddress", "hotelZipcode"},

(2) *subGoal*={"restaurantAddress"},

(3) *soln* = {`findHotel`, `findDirection`}.

- PD_{ws}("restaurantAddress")={`gudieRestaurant`, `findRestaurant`} and *wSpace*={`findRestaurant`, `guideRestaurant`}. Since each web service in *wSpace* has $h_{sg}(w)$=1 uniformly, we can randomly select χ={`findRestaurant`} and then, we can update as follows:

  (1) s=s∪{"hotelAddress", "hotelZipcode"},

  (2) *subGoal*=∅,

  (3) *soln* ={`findHotel`,`findDirection`, `findRestaurant`}.

- Since *subGoal*=∅, we stop the repetitions.

- Set $s = r_2^i$, print "`findHotel`", and then print ⇒. This is because $\Omega(s)$={`findHotel`}.

- Set s=s∪{"hotelAddress","hotelZipcode"}, print "`findRestaurant`", and then print ⇒. This can be explained because $\Omega(s)$={`findRestaurant`}.

- Set s=s∪{"restaurantAddress","restaurantRate"}, print "`findDirection`", and then print ⇒. This is because $\Omega(s)$={`findDirection`}.

- Since *soln*=∅, we stop the regression search with a solution: "`findHotel`" ⇒ "`findRestaurant`" ⇒ "`findDirection`".

We can prove the correctness of the forward search of WSPR by using the loop invariants technique [30].

Theorem 6.1.1 (Correctness of the forward search of WSPR)**.** *The forward search of WSPR runs on* $\Psi =< S, s_0, S_G, \Omega(.), f, c >$ *with* $c(w) = 1$ *for all* $w \in W$*, and it terminates with the realization of* $g_{r^i}(p)$ *for all parameters* $p \in S_G$*.*

Proof. We use the following loop invariant: At the start of each iteration of the 'while' loop of Lines 2-7 in Algorithm 1, we obtain $g_{r^i}(p)$ for each parameter $p \in w^o$, where $w \in \delta$.

It suffices to show that for each parameter $p \in w^o$ (where $w \in \delta$), we obtain $g_{r^i}(p)$ at the time when p is added to state s.

- **Initialization**: Initially, $s = \emptyset$, and so the invariant is trivially true.
- **Maintenance**: For the purpose of contradiction, let u be the first parameter for which we do not obtain $g_{r^i}(u)$ when u is added to state s. This assumption implies that another state occurs afterward and the state includes u , at which we can obtain $g_{r^i}(u)$. However, this is false because $c(w) = 1$ for all $w \in W$. In other words, if $s_{i+1} = f(w_i, s_i)$, $g_{r^i}(y)$=$g_{r^i}(x) + 1$, where $x \in (s_i \setminus s_{i-1})$ and $y \in (s_{i+1} \setminus s_i)$. Therefore, we obtain $g_{r^i}(p)$ at the time when p is first added to state s.
- **Termination**: The termination condition is $s \supseteq r^o$. This implies that $s \in S_G$. Thus, we have obtained $g_{r^i}(p)$ for all parameters $p \in S_G$.

□

Similarly, we can use the loop invariants technique to prove the correctness of the regression search of WSPR.

Theorem 6.1.2 (Correctness of the regression search of WSPR)**.** *The regression search of WSPR terminates after obtaining a set of web services to form a path from r^i to r^o.*

Proof. We use the following loop invariant: at the start of each iteration of the 'while' loop of Lines 2-6 in Algorithm 2, we obtain a set of web services, *soln*, that can temporarily form a sequence and be executed to form the path from *subGoal* to r^o. Then, it suffices to show that for the web service w which was added recently in *soln*, w can be invoked by *subGoal*. Once we show that *subGoal* can invoke w, we rely on the *subGoal* relation described at Line 6 in Algorithm 2 to show that the subsequent invocation holds at all times thereafter.

- **Initialization**: Initially, $subGoal = r^o$ and $soln = \emptyset$, making the invariant trivially true.
- **Maintenance**: Let A be a web service added to *soln* at t time step, where $t = 1, 2, \ldots, T(\geq \max_{p \in r^o} g_{r^i}(p) - 1)$. As soon as A is added to *soln*, $subGoal(t)$ is updated such that $subGoal(t) = [subGoal(t+1) \setminus (A^o \cup r^i)] \cup A^i$. That is, $subGoal(t)$ contains A^i, which is a set of preconditions necessary to invoke A.

Therefore, the updated $subGoal(t)$ can always invoke A. After invoking A, we can obtain new information state $s = subGoal(t) \cup (A^o \cup r^i)$. Note that r^i is always available throughout the planning process. Since $s \supseteq sugGoal(t+1)$, there is at least one web service whose input parameters are included in s as long as $soln \neq \emptyset$. In this manner, we can invoke all web service in $soln$ until $soln = \emptyset$.

- **Termination**: The termination condition is $subGoal \setminus r^i = \emptyset$. This implies that r^i satisfies $subGoal$. In the forward search of WSPR, we saw that there is a state s, such that $s \supseteq r^o$. This suggests that there is at least one path from r^i to r^o. Due to the existence of paths from r^i to r^o, there is at least one web service w belonging to $soln$ at termination, such that w can be invoked only by r^i.

□

We can use Equation 6.1 to drive the lower bound of the optimal cost of WSC solutions. Note that the invocation cost of a web service is assumed to be 1. Thus, the optimal cost of a WSC problem coincides with the minimum number of web services required to solve the WSC problem.

Theorem 6.1.3. *The lower bound for the optimal cost of achieving r^o from r^i is* $\max_{p \in r^o} g_{r^i}(p)$.

Proof. For a set of parameters A, let $G^*(A)$ be the optimal cost function to achieve A from r^i. Alternatively, this can be viewed as the optimal cost of achieving a state s where A holds. The equation characterizing the function G^* is

$$G^*(A) = \min_{<B,w> \in R(A)} [c(a) + G^*(B)] \tag{6.3}$$

where $G^*(A) = 0$, if $A \subseteq r^o$. $R(A)$ refers to the set of pairs $< B, w >$, such that B is the result of regressing A through a web service w (i.e., $B = A \setminus w^o$). $c(w)$ is the cost of invoking w. Let G be a function with the same domain as G^*. Let us write $G \leq G^*$, if $G(A) \leq G^*(A)$, for any set of parameters A. We try to yield G that are lower bounds on G^*, and this trial is regarded as "relaxation". With

these considerations in mind, we can form the following cost function:

$$g_{r^i}^{max}(A) = \max_{p \in A} g_{r^i}(p) \tag{6.4}$$

It is evident that $g_{r^i}^{max} \leq G^*$, as the cost of achieving a set of parameters cannot be lower than the cost of achieving each of the parameters in the set. Since r^o is a set of parameters, $g_{r^i}^{max}(r^o) \leq G^*(r^o)$. Thus, $\max_{p \in r^o} g_{r^i}(p)$ is the lower bound of the optimal cost of achieving r^o from r^i. □

From the perspective of heuristic design, $g_{r^i}^{max}$ is called an "admissible" heuristics as it never overestimates the true costs. However, it is questionable to believe that $g_{r^i}^{max}$ is a good estimator for G^*. In fact, $g_{r^i}^{max}$ is admissible, but it is less informative because it focuses only on the most difficult sub-goals, ignoring all the others [21, 50]. Nonetheless, the majority of AI planners that originated from Graphplan use a refined version of $g_{r^i}^{max}$ for their search heuristics. As mentioned in Chapter 3, the algorithms using an admissible heuristic are often called *Iterative Deepening A* algorithms* (IDA*). However, Graphplan and its variants are more efficient than general IDA* because of the planning graph constructed in the forward search step. In Chapter 7, we will use $g_{r^i}^{max}$ as a baseline to compare WSC algorithms.

6.2 A*-variant algorithms

In this section, we suggest two web-service composition algorithms based on A* algorithm: WS* and adaptive WS*. We compare WSPR with the two algorithms, as well as demonstrate the superiority of the two-step guided search of WSPR over the simple best-first search strategy adopted by the A* algorithm. Each A*-variant algorithm has a specific heuristics to form its cost function [94]. Since the performance of an A*-variant algorithm heavily depends on the quality of the heuristics, it is important to use the right heuristics to strike a good balance between accuracy and speed.

In the WSC problem, an A* algorithm can be captured as follows. Given a state and a set of candidate web services to visit next, one chooses the service with

the "smallest" $f(w)$. $f(w)$ can be defined as follows:

$$f(w) = h(w) + g(w) \tag{6.5}$$

where $g(w)$ is the number of web services that is already selected between r^i and the current state s. $h(w)$ is a heuristic to estimate the number of web services, from the current state s to the goal state, where r^o is realized. Note that we can ignore $h(w)$ and instead use an exhaustive search algorithm over the search space S. However, in that case, the computational complexity increases rapidly up to $O(2^{|P|})$ because $|S|$ is proportional to $2^{|P|}$.

6.2.1 WS* Algorithm

In this algorithm, we rely on the following "best first search" algorithm-based hypothesis:

Hypothesis-2: *Choosing a web service with a greater contribution to match the goal parameters earlier in the forward search helps reach the goal state faster.*

$$h(w) = \frac{1}{|(r^o \setminus s) \cap w^o|} \tag{6.6}$$

In Equation 6.6, the remaining parameters of r^o that are yet to be found are $(r^o \setminus s)$. Then, the intersection of this and w^o is a set of parameters that w contributes. Therefore, WS* algorithm favors the w whose contribution to find remaining parameters is the maximum (i.e., the smallest $h(w)$).

From the operational perspective of A* algorithm, OPEN list of the A* algorithm corresponds to δ, and CLOSE list of the A* algorithm corresponds to *soln* in Algorithm 3. For a better understanding of the behavior of WS*, we make an illustration with the "Atherton" motivating example. The procedure to run WS* is as follows:

- At the beginning, we know r_2^i={"hotelName", "hotelCity", "hotelState"} and run WS* to find r_2^o={"restaurantMap", "restaurantDirection"}

  1. s={"hotelName", "hotelCity", "hotelState"}

Input : r^i and r^o
Output: $soln$ /* $soln$ is a stack */
1 $s = (r^o \setminus r^i)$;
2 print r^i, "⇒";
3 **while** $\neg(s \supseteq r^o)$ **do**
4 $\quad \delta = \Omega(s) \setminus soln$;
5 $\quad \chi = \arg\max_{w \in \delta} f(w)$; /* a tie breaks at random */
6 $\quad$ Push(soln,χ);
7 $\quad s = s \cup \chi^o$;
8 **for** $w(\in soln)$ **do**
9 $\quad$ print Pop($soln$), "⇒" ;
10 print r^o ;

Algorithm 3: WS* Algorithm

- With the current information state s, we can obtain:
  1. $\Omega(s)$={ `findHotel`},
  2. $\delta = \Omega(s)$,
  3. $soln$ = [`findHotel`] because $|\delta| = 1$,
  4. $s = s\cup$ {"hotelAddress", "hotelZip"}.
- With the updated information state s, we can obtain:
  1. $\Omega(s)$={ `findHotel`, `findRestaurant`, `guideRestaurant`},
  2. δ = {`findRestaurant`, `guideRestaurant`} by $\delta = \Omega(s) \setminus soln$,
  3. $h(w) = \infty$ for $\forall w \in \delta$,
  4. χ = `findRestaurant` (chosen at random),
  5. $soln$ = [`findHotel`,`findRestaurant`],
  6. $s = s\cup$ {"restaurantName", "restaurantPhone", "restaurantAddress"}.
- With the updated information state s, we can also +obtain:
  1. $\Omega(s)$={ `findDirection`, `findHotel`, `findRestaurant`, `guideRestaurant`},
  2. δ = {`findDirection`, `guideRestaurant`} by $\delta = \Omega(s) \setminus soln$,
  3. h(`findDirection`)=1/2 while h(`guideRestaurant`)=∞,

4. χ = `findDirection`,
5. $soln$ = [`findHotel`,`findRestaurant`,`findDirection`],
6. $s = s \cup$ {"restaurantMap", "restaurantDirection"}.

- $s \supseteq r_2^o$ and we stop the procedure.

6.2.2 Adaptive WS* Algorithm

It is not uncommon for the WS* algorithm to have multiple candidate web services with the same $h(w)$ value in OPEN list. Then, instead of picking one arbitrarily, the adaptive WS* tries to "look ahead" to adaptively determine the best one. That is, in addition to considering $h(w)$, one may look at the $h(w')$ where $w' \in \delta'$ and $\delta' = [\Omega(s \cup w^o) \setminus \{w\}] \setminus soln$. Therefore, adaptive WS* chooses a web service such that the combined contribution of itself and its child is the maximum. This can be captured with the following modified heuristic function:

$$h(w) = \frac{1}{|(r^o - s) \cap (w^o \cup [h_c(w)]^o)|} \quad (6.7)$$

$$h_c(w) = \arg\max_{w' \in \delta'} |(r^o \setminus (s \cup w^o)) \cap w'^o| \quad (6.8)$$

In general, the adaptive WS* algorithm requires more recourses (e.g., time or memory) but produces more accurate solution - thanks to the better informed heuristics. This will be experimentally validated in the next section.

Similar to WS*, we can illustrate the adaptive WS* algorithm with the Atherton motivating example. The majority of the procedure is the same as the WS* algorithm, but the main difference can be found in calculating $h(w)$. Assume that we are with s ={"hotelName", "hotelCity", "hotelState", "hotelAddress", "hotelZip"} and δ={`findRestaurant`, `guideRestaurant`}. Now, we have to choose χ by calculating $h(w)$, where $w \in \delta$. Remember that in the WS*, both `findRestaurant` and `guideRestaurant` had ∞ as the value of $h(w)$, and therefore one of them was chosen at random. On the contrary, in a tie situation, the adaptive WS* forces `findRestaurant` and `guideRestaurant` to check their child nodes. As a result, `findRestaurant` has 1/2 as the value of $h(w)$ because δ'={ `findDirection`}, and

`findDirection` has the contribution of two. As a result, the combined contribution of `findRestaurant` and its child `findDirection` is bigger than the combined contribution of `guideRestaurant` and its child, and `findRestaurnat` is chosen.

6.3 Comparison of WSPR and A*-variants

In this section, we compare A*-variants (WS* and Adaptive WS*) and WSPR. Note that WSPR can be considered to be a guided search because it uses $g_{r^i}(p)$ and $PD_{ws}(p)$ obtained from the first step . On the contrary, WS* and Adaptive WS* can be classified into unguided searches because their heuristics can become useless if the distance between the goal state r^o and the initial state r^i is increased.

To compare the efficiency of the three proposed algorithms, we conducted experiments using the EEE05 test set. The EEE05 test set is a synthetic test set that contains artificially created composition scenarios, and the test set and test requests appear to be manually created by human experts. For example, the test request No. 15 has the following input and output sets

- Input set = {"pickupLocationName", "pickupLocationID", "firstName", "lastName", "middleInitial", "custStreetAddress", "custCityAddress", "custStateAddress", "custZipAddress"}
- Output set = {"shipmentTrackingNumber", "shipmentCost"}

The test set size is just 100. Although the test set is small, the EEE05 test set is still challenging because it is not simple for humans to solve them optimally in a short time. Note that the EEE05 contest originally offered 15 test requests, but six test requests (i.e., No. 4, 6, 7, 9, 11, and 12) out of 15 are discarded since there are syntax errors in the requests[3].

Throughout the experiments, we used the following evaluation metrics:

1. *Time*: it measures how long an algorithm takes to find a solution in milliseconds. This is a measure of computational efficiency.

[3]In the EEE05 web service composition contest, generating the test set and requests were conduced manually, leaving room for human error. In fact, six requests turned out having no solutions due to syntax errors occurred by the error-prone manual process of data generation. Since we were not able to fix the errors by ourselves, we discarded the problematic requests

Table 6.1: Comparison of three algorithms in terms of $\#W$ and $Time$

(Test ID)	WS*		Adaptive WS*		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
1(1)	**4**	0.156	5	1.704	**4**	**0.01**
2(2)	6	0.016	5	16	**4**	**0.01**
3(3)	**3**	**0.01**	**3**	**0.01**	**3**	**0.01**
4(5)	**1**	0.016	**1**	0.032	**1**	**0.01**
5(8)	3	0.125	**2**	1.141	**2**	**0.01**
6(10)	**1**	0.015	**1**	**0.01**	**1**	**0.01**
7(13)	10	0.562	**2**	0.875	**2**	**0.01**
8(14)	10	0.672	**2**	0.703	**2**	**0.01**
9(15)	**1**	0.015	**1**	0.047	**1**	**0.01**

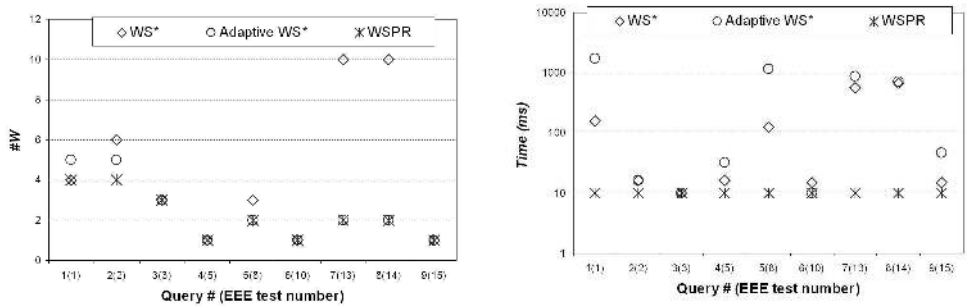


Figure 6.1: Comparison of three algorithms (left) $\#W$ (right) $Time$

2. $\#W$: the number of web services in a solution. This gives the quality of the solution obtained, or the effectiveness of the solution.

The smaller both values get, the better the solutions are. All experiments were performed on a PC with the Pentium 4 with 523Mb RAM, running Windows XP at 1.7GHz. All algorithms were implemented in Python 2.3. The results of the experiments on the EEE05 test set are shown in Table 6.1 and Figure 8.4.

All competitors have no difficulty in solving the problems. However, in terms of $Time$ and $\#W$, WSPR outruns the other two A*-based algorithms. Regarding two A*-variants, the adaptive WS* shows a better performance than WS* in terms of accuracy ($\#W$). However, the efficiency ($Time$) is worse than WS* as a whole. These results are expected because the adaptive WS* builds a much bigger OPEN list, due to the look ahead process. It is evident that a bigger OPEN list can yield a

greater chance to find accurate solutions, whereas it takes a longer computational time than the WS*. Consequentially, WS* is better than the the adaptive WS* in terms of $Time$, while the adaptive WS* is superior to the WS* in terms of $\#W$.

In these experiments, we see that WSPR is uniformly better than the two A*-based algorithms for all of the experiments. In other words, WSPR has both an efficiency and accuracy advantage by leveraging on $g_{r^i}(p)$ and $PD_{ws}(p)$ that are obtained from the first step as guidance. The WS* and adaptive WS* have a clear problem with their heuristics. As confirmed by the experimental evaluation, both heuristics are not able to guide the search, if the goals are far from the initially given information. Consequently, we can argue that the two step approach is superior to the best-first forward search algorithm, like the A* algorithm, especially when solutions require longer paths from the initial state r^i to the goal state r^o. In the following chapters, we concentrate on comparing WSPR with other AI planners, which also have same two-step structure but use different heuristics.

Chapter 7

Experimental Validation

In this chapter, we compare the performance of WSPR and other prominent AI planners (Blackbox 4.2, IPP 4.1, and Graphplan) in terms of effectiveness and computational efficiency. In addition, we investigate the scalability of WSPR with respect to increasing the test set size. We also study the robustness of WSPR in the presence of diverse test sets and composition scenarios.

To validate an algorithm for the WSC problem, one needs both test sets and test requests. We prepared three types of test sets[1] as follows:

1. EEE05 test set: human-generated test sets that are small-scale with only 100 web services but with non-trivial test requests.

2. ICEBE05 test sets: synthetically generated large-scale test sets.

3. WSBen-generated test sets: $baTS$, $nwsTS$, and $erTS$ generated in Chapter 5. They are synthetically generated large-scale test sets featured with diverse underlying network topologies.

In the experiments described below, we compare the performance of WSPR and three prominent AI planners: Graphplan, Blackbox and IPP.

As explained in Chapter 3, Blackbox and IPP are extended planning systems that originated from Graphplan. In particular, Blackbox is extended to be able

[1]We could have generated test sets using real 1,544 web services that we had gathered in Chapter 4. However, 1,544 web services were not large enough for us and the lack of correlation among real web services (e.g., the network diameter of G_{op} is just five) makes it hard to generate "challenging" test sets.

to map a plan graph into a set of clauses for which Blackbox forms a satisfiability problem (SAT problem). For the SAT problem, Blackbox applies the local-search SAT solver, Walksat, so that Blackbox can run even with a large number of operators. Throughout the experiments, we use two evaluation metrics as we did in Chapter 6:

1. $Time$: it measures how long an algorithm takes to find a solution in milliseconds. This is a measure of the computational efficiency.

2. $\#W$: the number of web services in a solution. This gives the quality of the solution obtained, or the effectiveness of the solution.

The smaller both values get, the better the solutions are. In other words, algorithms that take less running time while producing right solutions that use fewer web services are considered to be "good". Note that all AI planners (Blackbox, IPP, and Graphplan) are all optimal parallel planners that minimize the number of time steps, but not necessarily the number of actions (i.e., the number of web services). All AI planners run with their default options, except that the maximum number of nodes for Blackbox and Graphplan was set to 32,768 and 10,000, respectively. Commonly, the time to read the operator and fact files is not included in $Time$ measurement. Blackbox and IPP accept only the PDDL format, while Graphplan accepts only the STRIPS format for their operator and fact files. Note that an operator file corresponds to a test set, and a fact file corresponds to a test request file. WSBen provides a function to convert test sets and requests into PDDL and STRIPS files automatically. The experiments were performed on Linux with three Intel® Xeon^{TM} CPU, running at 2.4GHz with 8Gb RAM. We also assess the performance difference brought by WSPR heuristics. To this end, we include WSPR without the heuristics, named as "WSPR w/o heuristics," in the list of competitors.

7.1 EEE05 test set

The first set of experiments deals with the EEE05 contest set. Further information about the EEE05 test set can be found in Section 6.3. In this experiment, all competitors except Graphplan, solved the optimal solutions for all test queries

within 0.1 seconds as shown in Table 7.1. The problem of Graphplan is caused by the conflicting two open- and closed-world assumptions.

The closed world assumption is the presumption that what is not currently known to be true is false. The opposite of the closed world assumption is the open world assumption. In the open world assumption, what is not stated is currently unknown. In general, the open world assumption is much preferred in many modern systems such as RDF and OWL. However, SQL and XML systems adopt the closed world assumption. Therefore, RDF and OWL consider that every tuple not explicitly contained in the semantic web or ontology is implicitly assumed to represent a fact that is unknown rather than false; SQL and XML consider these tuples to be false. For a simple example, suppose that we have a statement: `Mary is a citizen of France`, and we are asked by a question: `Is Mary a citizen of Canada?`. According to the closed world assumption, the answer is 'No'. On the contrary, under the open world assumption, the answer is 'Unknown' (Mary could have dual citizenship).

In the closed world assumption, web services must have a rule: $r^o \supset r^i$. Otherwise, parameters p which is in r^i but not in r^o becomes false. On the contrary, in the open world assumption, all parameters r^i hold true all the way through the planning process once the web service containing r^i is invoked. In the web services context, the closed world assumption is not appropriate because most web service does not have input parameters duplicated into output parameters. Therefore, we applied the open world assumption for all planners.

However, under the open world assumption, Graphplan makes an assertion error when it discovers web services w, such that $w^i \cap w^o \neq \emptyset$ due to its strict internal reasoning system. It is true that the EEE05 test set has several web services w such that $w^i \cap w^o \neq \emptyset$. For example, a web service named `findWeather` has an input parameter set {"forecastDate", "forecastCityAddress"} and an output parameter set {"forecastDate", "forecastCityAddress", "forecastTemparature"}, with "forecastDate" and "forecastCityAddress" occurring in both sets. While Graphplan cannot handle re-occurring parameter p that is $p \in (w^i \cap w^o)$ under the open world assumption, other AI planners can handle this abnormal situation intelligently.

The result of the experiments on EEE05 test set is shown in Table 7.1. Note that EEE05 contest originally offered 15 test requests, but six test requests (Test

Table 7.1: Results of the EEE05 test set

test ID	Blackbox		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
1	**4**	0.04	**4**	0.11	**4**	**0.01**
2	**4**	0.04	**4**	0.12	**4**	**0.01**
3	**6**	0.07	**6**	0.21	**6**	**0.01**
5	**1**	**0.01**	**1**	0.06	**1**	**0.01**
8	**2**	**0.01**	**2**	0.08	**2**	**0.01**
10	**1**	**0.01**	**1**	0.06	**1**	**0.01**
13	**2**	**0.01**	**2**	0.08	**2**	**0.01**
14	**2**	**0.01**	**2**	0.08	**2**	**0.01**
15	**1**	**0.01**	**1**	0.07	**1**	**0.01**

Number. 4, 6, 7, 9, 11, and 12) out of 15 are discarded since there are syntax errors in the requests. The three competitors had no obstacles to generate the optimal solution. In terms of $\#W$, the three competitors have the same values. In terms of $Time$, WSPR outperforms others but the differences between competitors are insignificant, suggesting that the EEE05 test set is simple.

7.2 ICEBE05 test set

The second set of experiments deals with the ICEBE05 contest set. ICEBE05 provides 18 test sets with their complexities varied in different dimensions[2] [83]. Note that all test requests of ICEBE05 can be solved by the full-matching operation, so that $\#W = \max_{p \in r^o} g_{r^i}(p)$[3]. We found that "Composition2-100-32" was the most difficult test set among 18 test sets, as discussed in Appendix B. Note that we converted ICEBE05 test sets into PDDL and STRIPS file format to feed AI planners through WSBen, which provides the conversion function. The results of the experiments with "Composition2-100-32" are shown in Table 7.2. As shown in the table, all competitors, except Graphplan, had no problem in solving the optimal solutions for all test requests within a reasonable time. We found that Graphplan fails to read the operation file when the number of operations in the

[2]See Appendix B for the analysis result of the computational complexity of ICEBE05 test sets.

[3]It is the lower bound of the WSC solution. See Theorem 6.1.3.

Table 7.2: Results of the Composition2-100-32 test set of ICEBE05

test ID	Blackbox		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
1	**6**	**4.36**	**6**	11.66	**6**	4.92
2	**6**	**4.35**	**6**	11.55	**6**	4.86
3	**6**	5.81	**6**	12.73	**6**	**4.92**
4	**7**	**5.8**	**7**	12.76	**7**	6.06
5	**7**	**5.8**	**7**	12.75	**7**	6.07
6	**7**	**5.86**	**7**	12.83	**7**	6.02
7	**8**	**7.25**	**8**	13.81	**8**	7.31
8	**8**	7.34	**8**	13.85	**8**	**7.24**
9	**8**	**6.97**	**8**	13.93	**8**	7.18
10	**8**	**7.04**	**8**	13.96	**8**	7.26
11	**1**	3.42	**1**	11.86	**1**	**0.68**

file exceeds 5,000; "Composition2-100-32" has 8,356 web services (or operations, in the PDDL and STRIPS operation files). Regarding $\#W$, all competitors had no problem to generate the minimal length solution to each of the test requests. In terms of $Time$, Blackbox and WSPR shows similar results, while IPP shows the worst performance which is due to the high overhead of its internal heuristic algorithm.

7.3 Test sets generated by WSBen

The third set of experiments deals with $baTS$, $nwsTS$ and $erTS$. As described in Section 5.2, we build three test set frameworks by specifying the xTS of WSBen as follows:

(1) $baTS = \langle$ 100,Barabasi-Albert(100,6),0.8,5,$|W|$ $\rangle$

(2) $nwsTS = \langle$100,Newman-Watts-Strogatz(100,6,0.1),0.8,5,$|W|\rangle$

(3) $erTS = \langle$100,Erdos-Renyi(100,0.06),0.8,5,$|W|\rangle$

Each domain of $baTS$, $nwsTS$, and $erTS$ has seven different sizes of test sets by varying $|W|$ as 1,000, 3,000, 5,000, 10,000, 20,000, 30,000, and 50,000. Consequently, 21 test sets were prepared. For each of 21 test sets, we generated five test

Table 7.3: Results of baTS with $|W| = 1000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	**4**	5.92	**4**	**0.01**	**4**	9.800	**4**	0.062
r_2	**2**	4.15	**2**	**0.01**	**2**	9.74	**2**	0.078
r_3	**7**	67.141	**7**	**0.01**	**7**	10.56	**7**	0.035
r_4	**35**	445.19	37	**0.17**	**35**	23.02	**35**	3.35
r_5	**4**	0.59	**4**	**0.01**	**4**	9.72	**4**	0.062

requests. Each test request has a different solution length so that we can compare competitors with different test scenarios (i.e., short versus long solution length). In order to create a test requests r, we used WSBen, which is designed to generate test requests automatically. For this purpose, WSBen operates as follows:

1. WSBen selects a Cluster $j \in G_{cl}$ at random.

2. WSBen copies all parameters in the Cluster j (i.e., Pa_j) into r^i, and then r^o is constructed so that it consists of the first five largest parameters in terms of $g_{r^i}(p)$. Consequently, parameters in r^o are farthest away from parameters in r^i in a parameter space.

WSBen repeats the above procedure five times, resulting in generating five requests for each test set.

7.3.1 Comparison results over $baTS$

The results of the five test requests of $baTS$ with $|W| = 1,000$ are shown in Table 7.3. The AI planners and WSPR have no difficulty in computing solutions with a small $\#W$. On the other hand, IPP solves the fourth request by using two more web services than the others. Regarding $Time$, Graphplan shows the best performance, while Blackbox and IPP show poor performance. Overall, $baTS$ with $|W| = 1,000$ is proved not to be a challenging test set for most of the planners because the performance differences between competitors are not significant.

The results over the five test requests of $baTS$ with $|W| = 3,000$ are shown in Table 7.4. While WSPR has no trouble in finding solutions, AI planners fail to solve some cases. Blackbox, IPP, and Graphplan fail to solve the second request.

Table 7.4: Results of baTS with $|W| = 3,000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	61	478.695	-	-	-	-	**37**	**11.75**
r_2	-	-	-	-	-	-	**66**	**16.78**
r_3	**5**	5	**5**	**0.09**	**5**	26.22	**5**	24.453
r_4	**9**	27.78	**9**	**0.11**	**9**	28.56	**9**	2.626
r_5	**4**	1.4	**4**	**0.04**	**4**	23.97	**4**	0.75

Table 7.5: Results of baTS with $|W| = 5,000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	-	-	-	-	-	-	**59**	**22.358**
r_2	-	-	-	-	-	-	**50**	**24.921**
r_3	-	-	-	-	-	-	**68**	**20.062**
r_4	69	609.326	-	-	-	-	**45**	**19.53**
r_5	-	-	-	-	-	-	**83**	**29.125**

Additionally, IPP and Graphplan also fail to solve the first request. Regarding $Time$, Graphplan demonstrates the best performance in solving the last three requests.

Table 7.5 shows the results of the five test requests of $baTS$ with $|W| = 5,000$. Graphplan and IPP run out of memory for all cases. Blackbox also fails except for the fourth request, but the solution length of the fourth request is longer than that of WSPR. WSPR finds all solutions without difficulty. Regarding $Time$, WSPR solves all requests within 30 seconds, but Blackbox takes 609 seconds to solve the fourth request. The experiment results of $baTS$ with $|W| = 5,000$ implies that the comparison of AI planners and WSPR is in vain once the number of web services exceeds 5,000. Judging from the results above, the WSPR heuristics with the strategy in support of locating a fully-matching web service first in a tie situation, is in effect when the underlying network topology follows the scale-free network topology.

Table 7.6: Results of nwsTS with $|W| = 1,000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	**2**	0.1	**2**	**0.01**	**2**	0.03	**2**	0.062
r_2	33	69.307	33	**0.78**	49	3.26	**27**	1.687
r_3	**28**	5.990	**28**	**0.62**	35	2.9	34	1.437
r_4	33	9.130	34	**0.81**	49	3	**27**	1.781
r_5	56	1.800	56	**1.02**	-	-	**24**	1.312

Table 7.7: Results of nwsTS with $|W| = 3,000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	48	571.633	-	-	48	29.52	**17**	**7.812**
r_2	35	114.678	-	-	35	28.57	**18**	**7.687**
r_3	24	192.997	-	-	24	30.19	**19**	**7.375**
r_4	26	11.884	-	-	26	28.39	**16**	**6.515**
r_5	**31**	111.21	-	-	-	-	**31**	**7.352**

7.3.2 Comparison results over $newTS$

The results of the five test requests of $nwsTS$ with $|W| = 1,000$ are shown in Table 7.6. The AI planners and WSPR have no difficulty in generating solutions, but IPP fails to solve the fifth request. Regarding $\#W$, each competitor shows various results. As a whole, WSPR presents better solution, except for the fourth request. As a result, it can be said that WSPR shows better solutions in 80% of the cases in terms of $\#W$. Regarding $Time$, Graphplan shows the best performance, while Blackbox and IPP show poor performance.

The results over the five test requests of $nwsTS$ with $|W| = 3,000$ are shown in Table 7.7. Graphplan runs out of memory for all cases. IPP also fails to solve the fifth request. WSPR overruns other planners in both $\#W$ and $Time$. Note that $Time$ values in this domain are generally longer than those of $baTS$. This is because that the average network diameter of $nwsTS$ is longer than that of $baTS$.

Table 7.8 shows the results of the five test requests of $nwsTS$ with $|W| = 5,000$. Graphplan runs out of memory for all cases. IPP also fails to solve the fifth request. WSPR finds all solutions, and four out of the five solutions are better than the other planners. Regarding $Time$, WSPR overruns Blackbox and IPP. Compared

Table 7.8: Results of nwsTS with $|W| = 5,000$. Time in second

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	47	61.96	-	-	47	22.02	**30**	**7.875**
r_2	**29**	58.4	-	-	-	-	40	**8.718**
r_3	39	30.98	-	-	39	21.42	**28**	**7.828**
r_4	38	7.14	-	-	38	21.6	**26**	**7.015**
r_5	39	3.54	-	-	39	20.33	**26**	**6.577**

Table 7.9: Results of erTS with $|W| = 1,000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	41	40.54	41	1.67	41	7.67	**39**	**1.469**
r_2	**12**	0.46	**12**	**0.01**	**12**	1.58	**12**	0.187
r_3	**1**	0.2	**1**	**0.04**	**1**	1.42	**1**	0.077
r_4	74	193.539	-	-	-	-	**59**	**2.219**
r_5	55	7.630	55	5.6	-	-	**38**	**1.421**

to $baTS$, Blackbox and IPP can still run even though $\#W$ and $Time$ are poorer than WSPR. According to the results above, the WSPR heuristics is in effect when the underlying network topology follows the small-world network.

7.3.3 Comparison results over $erTS$

The results of the five test requests of $erTS$ with $|W| = 1,000$ are shown in Table 7.9. Graphplan fails to solve the fourth request and IPP fails to solve the fourth and fifth requests. WSPR and Blackbox have no problem in computing solutions, but WSPR overruns Blackbox in terms of $\#W$ and $Time$.

Table 7.10 shows the results of the five test requests of $erTS$ with $|W| = 3,000$. Graphplan fails in all cases and IPP fails to solve the first and second requests. WSPR and Blackbox have no trouble in finding the solutions, but WSPR overruns Blackbox in terms of $\#W$, except for the third case. Regarding $Time$, WSPR demonstrates better performance than others in three cases.

Table 7.11 shows the results of the five test requests of $erTS$ with $|W| = 5,000$. Graphplan fails in all cases and IPP fails to solve the fifth request. WSPR and Blackbox have no problem in obtaining solutions, but WSPR overruns Blackbox

Table 7.10: Results of erTS with $|W| = 3,000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	75	38.09	-	-	-	-	**56**	**5.703**
r_2	**50**	16.02	-	-	-	-	56	**4.858**
r_3	22	18.68	-	-	22	24.78	**19**	**4.766**
r_4	23	**4.38**	-	-	23	21.06	**22**	4.672
r_5	38	**4.01**	-	-	38	21	**26**	4.625

Table 7.11: Results of erTS with $|W| = 5,000$

test requests	Blackbox		Graphplan		IPP		WSPR	
	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$	$\#W$	$Time$
r_1	43	**7.85**	-	-	43	44.21	**19**	13.719
r_2	24	96.111	-	-	24	47.46	**13**	**9.094**
r_3	42	95.63	-	-	42	48.47	**15**	**8.202**
r_4	48	80.592	-	-	48	47.75	**22**	**8.984**
r_5	17	**8.5**	-	-	-	-	**12**	8.625

in terms of $\#W$. Regarding $Time$, WSPR shows better performances than others in three cases. From the results, we can infer that the WSPR heuristics is in effect even when the underlying network topology follows the random network.

In the experiments above, we assessed the performance of WSPR in comparison with three AI planners: Graphplan, Blackbox, IPP. As a whole, WSPR presents better solutions in 80% of the cases in terms of $\#W$ and also shows competitive results in $Time$ performance. On the contrary, AI planners show unstable performances or failures as the test size increases. In particular, most AI planners do not operate at all when the underlying web-service network follows the scale-free network topology and the test size exceeds 5,000. One of the reasons for the failure of AI planners is that the heuristics implicitly represented by the plan graph is a very poor estimator in the web-service domain. Here, sub-goals are mostly independent, like in the Gripper domain[4] [70]. As a result, Graphplan based planners, such as IPP, that perform a form of IDA* search must perform many iterations before finding a solution. Even though Blackbox is an AI planner based on Graphplan, Blackbox can run a large size where IPP and Graphplan cannot operate. It is ev-

[4]See Section 3.3.1 for the explanation of the Gripper domain

ident because Blackbox uses the local search SAT solver, Walksat[5]. However, the results show that WSPR maintains a better performance than Blackbox in terms of $\#W$. The comparison between Blackbox and WSPR will be further analyzed in Section 7.3.5.

From the above experiments using diverse test sets, we can understand how different network models of G_{cl} influences the performance of WSC algorithms. In general, given the same number of clusters, the Barabasi-Albert model generates G_{cl} with a greater number of parameters and a larger variance of the number of parameters between clusters than the Newman-Watts-Strogatz and Erdos-Renyi models do. Due to the greater number of parameters and larger variance, $baTS$ based on the Barabasi-Albert model needs more partial-matching web services to fulfill the given requests than others. As such, the increasing need for partial-matching web services leads to increasing $\#W$ and $\mathcal{T}$. This is the reason three AI planners almost failed to run in the $baTS$ case.

7.3.4 Scalability of WSPR

We change our focus from the comparison of WSPR and AI planners to the assessment of scalability of WSPR. From Table 7.12 to Table 7.18, we list the comparison results of WSPR and WSPR w/o heuristics over nine test sets in $baTS$ domain.

As shown in Table 7.12, both algorithms have no complications in computing solutions in the case of $baTS$ at $|W|$=1,000. On the other hand, WSPR w/o heuristics takes one more web service to solve the fourth request. Regarding $Time$, they both take a similar amount. Therefore, the time to compute the heuristics is trivial in this case.

Table 7.13 shows that both algorithms have no difficulty in computing solutions when $|W|$=3,000. As for $\#W$, WSPR overruns WSPR w/o heuristics. Regarding $Time$, they both are almost the same, except that WSPR takes twenty times longer in the third test request than WSPR w/o heuristics. This implies we can face intractable tie situations where large number of web services have the same heuristic value, thereby requiring considerable computational time to break the tie.

[5]See Section 3.3 for the explanation of the Walksat

Table 7.12: Results of baTS with $|W| = 1,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**4**	**0.062**	**4**	0.078	1
r_2	**2**	**0.078**	**2**	**0.078**	1
r_3	**7**	**0.035**	**7**	0.39	4
r_4	**35**	**3.35**	36	3.437	7
r_5	**4**	**0.062**	**4**	0.078	1

Table 7.13: Results of baTS with $|W| = 3,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**37**	**11.75**	51	11.671	7
r_2	**66**	**16.78**	82	17.233	7
r_3	**5**	24.453	**5**	**1.39**	4
r_4	**9**	2.626	11	**2.562**	5
r_5	**4**	**0.75**	5	**0.75**	3

Table 7.14: Results of baTS with $|W| = 5,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**59**	**22.358**	89	22.626	7
r_2	**50**	24.921	82	**24.405**	8
r_3	**68**	**20.062**	100	23.766	7
r_4	**45**	19.53	63	**19.23**	6
r_5	**83**	29.125	126	**28.724**	9

The results of the five test requests of *baTS* with $|W| = 5,000$ are shown in Table 7.14. Both algorithms have no obstacles in calculating solutions. However, similar to the *baTS* with $|W| = 1,000$ case, WSPR overruns WSPR w/o heuristics. Regarding $Time$, both yield similar results.

As shown in Table 7.15, both algorithms have no trouble in computing solutions of the five test requests of *baTS* with $|W| = 10,000$. WSPR overruns WSPR w/o heuristics in terms of $\#W$ and $Time$. It is interesting to see that WSPR is rather reduced in $\#W$ as compared to the previous small size test sets. We can infer that as the number of web services increases, the chance to match web services fully increases accordingly.

Table 7.15: Results of baTS with $|W| = 10,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**14**	**34.967**	40	35.469	5
r_2	**33**	**35.671**	59	44.312	5
r_3	**12**	**31.578**	53	31.671	4
r_4	**19**	**25.032**	42	26.219	4
r_5	**37**	**33.687**	51	34.937	6

Table 7.16: Results of baTS with $|W| = 20,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**23**	**69.109**	60	69.5	5
r_2	**20**	**61.453**	109	62.89	5
r_3	**14**	**69.437**	33	70.014	5
r_4	**22**	77.218	73	**74.125**	5
r_5	**19**	**56.187**	71	58.875	4

Table 7.16 displays that both algorithms have no barrier in finding solutions in $baTS$ with $|W| = 20,000$. Like in the previous cases, WSPR overruns WSPR w/o heuristics in terms of $\#W$. Regarding $Time$, they both have similar results. Interestingly, as a whole, $Time$ in $|W| = 20,000$ takes two times longer than that reported in $|W| = 10,000$. This result indicates that WSPR and WSPR w/o heuristics are polynomial algorithms proportional to $|W|$.

The results of the five test requests of $baTS$ with $|W| = 30,000$ are shown in Table 7.17. Like in previous cases, both algorithms do not have a problem in computing solutions, but WSPR overruns WSPR w/o heuristics in terms of $\#W$. It is interesting to view that in the first and second requests, $\#W$ of WSPR reaches the lower bound of optimal $\#W$. Since optimal values cannot be lower than those bounds[6], it is evident that WSPR finds the optimal solutions in these two cases. Regarding $Time$, both have similar results. As a whole, $Time$ is three times longer than that reported in $|W| = 10,000$. Thus, it can be proved experimentally that WSPR and WSPR w/o heuristics are polynomial algorithms proportional to $|W|$.

As shown in Table 7.18, like in the previous cases, both algorithms have no

[6]See Theorem 6.1.3

Table 7.17: Results of baTS with $|W| = 30,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**4**	77.657	64	**73.483**	4
r_2	**4**	**74.5**	36	83.703	4
r_3	**6**	**94.828**	26	133.343	4
r_4	**17**	80.297	61	**78.139**	4
r_5	**12**	79.391	21	**76.78**	4

Table 7.18: Results of baTS with $|W| = 50,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**12**	**121.891**	31	242.516	4
r_2	**20**	**172.375**	76	179.875	6
r_3	**13**	**118.467**	32	123.187	4
r_4	**8**	**155.375**	52	162.718	5
r_5	**13**	**115.141**	53	115.78	4

difficulty in computing solutions, and WSPR overruns WSPR w/o heuristics in terms of both $\#W$ and $Time$. It is interesting to observe that the $Time$ of WSPR w/o heuristics at the first request takes twice as long as that of WSPR. It suggests that the efficiency brought by the heuristics of WSPR becomes more distinctive with the increase of $|W|$. As a whole, both algorithms are approximately five times slower in $Time$ as compared to $Time$ when $|W|$=10,000. Accordingly, it can be further proved experimentally that WSPR and WSPR w/o heuristics are polynomial algorithms proportional to $|W|$.

From Table 7.19 to Table 7.25, we list the comparison results of WSPR and WSPR w/o heuristics for the nine test sets in $nwsTS$ domain.

The results of the five test requests of $nwsTS$ with $|W| = 1,000$ are shown in Table 7.19. Both algorithms do not have a problem in computing solutions. However, WSPR overruns WSPR w/o heuristics, as in $baTS$ domain. Regarding $Time$, they both are approximately the same.

As shown in Table 7.20, both algorithms have no trouble in computing solutions in $|W| = 3,000$. Like in previous cases, WSPR overruns WSPR w/o heuristics, and regarding $Time$, both are almost the same. This means that WSPR computes

Table 7.19: Results of nwsTS with $|W| = 1,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**2**	0.062	2	**0.047**	1
r_2	**27**	1.687	44	**1.609**	15
r_3	**34**	1.437	39	**1.391**	12
r_4	**27**	1.781	42	**1.702**	15
r_5	**24**	1.312	30	**1.234**	10

Table 7.20: Results of nwsTS with $|W| = 3,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**17**	**7.812**	24	8.733	9
r_2	**18**	**7.687**	34	7.894	9
r_3	**19**	7.375	43	**7**	8
r_4	**16**	6.515	26	**6.171**	7
r_5	**31**	**7.352**	55	7.5	9

Table 7.21: Results of nwsTS with $|W| = 5,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**30**	7.875	64	**7.562**	11
r_2	**40**	8.718	54	**7.875**	12
r_3	**28**	7.828	47	**7.5**	11
r_4	**26**	7.015	43	**6.78**	10
r_5	**26**	6.577	43	**6.28**	9

its heuristics in regression search without loss of computation time.

Table 7.21 shows that both algorithms have no difficulty in calculating solutions when $|W| = 5,000$, but WSPR overruns WSPR w/o heuristics. Regarding $Time$, both are almost same.

The results of the five test requests of $nwsTS$ with $|W| = 10,000$ are shown in Table 7.22. The results coincide with previous results in that both algorithms have no obstacles in computing solutions, but WSPR overruns WSPR w/o heuristics in terms of $\#W$ and $Time$. It is interesting to see that in the first, fourth, and fifth requests, WSPR finds optimal solutions.

The results of the five test requests of $nwsTS$ with $|W| = 20,000$ are disclosed

Table 7.22: Results of nwsTS with $|W| = 10,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**13**	**17.485**	35	17.719	13
r_2	**31**	18.782	76	**18.233**	14
r_3	**32**	19.906	83	**19.343**	15
r_4	**8**	11.875	18	**11.062**	8
r_5	**15**	20.062	73	**18.53**	15

Table 7.23: Results of nwsTS with $|W| = 20,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**10**	27.03	21	**26.562**	10
r_2	**8**	22.671	18	**21.25**	8
r_3	**11**	**25.812**	31	25.953	11
r_4	**9**	**21.842**	30	22.062	9
r_5	**10**	25.78	31	**24.358**	10

in Table 7.23. Both algorithms have no difficulty in computing the solutions. Similar to the previous cases, WSPR overruns WSPR w/o heuristics in terms of $\#W$. Regarding $Time$, both have similar results. Interestingly, WSPR finds the optimal solution for all cases. It highlights the superiority of WSPR in terms of its effectiveness.

As displayed in in Table 7.24, both algorithms have no difficulty in computing solutions. Similar to the previous cases, WSPR overruns WSPR w/o heuristics in terms of $\#W$. Regarding $Time$, both have similar results. WSPR finds the optimal solution in the second and fourth cases. Note that $\#W$ of WSPR is roughly three times less than that for WSPR w/o heuristics. Interestingly, as a whole, both algorithms become three times slower in $Time$ than in $|W| = 10,000$. Thus, it can be proved experimentally that WSPR and WSPR w/o heuristics are polynomial algorithms proportional to $|W|$ in this domain.

Table 7.25 shows the results of the five test requests of $nwsTS$ with $|W| = 50,000$. The results are similar to previous cases. WSPR overruns WSPR w/o heuristics in terms of both $\#W$ and $Time$. In the second and fifth requests, WSPR finds optimal solutions. Be aware that WSPR is roughly five times slower

Table 7.24: Results of nwsTS with $|W| = 30,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**12**	**37.625**	30	38.766	10
r_2	**10**	**36.719**	70	38.171	10
r_3	**15**	**36.531**	45	37.717	11
r_4	**10**	40.985	33	**36.5**	10
r_5	**17**	50.672	55	**45.296**	13

Table 7.25: Results of nwsTS with $|W| = 50,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**22**	78.108	52	**74.655**	11
r_2	**7**	67.78	15	**60.468**	7
r_3	**9**	61.703	17	**56.483**	8
r_4	**8**	**53.812**	15	61.766	7
r_5	**10**	**81.812**	29	82.094	10

Table 7.26: Results of $erTS$ with $|W| = 1,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**39**	1.469	45	**1.453**	8
r_2	**12**	0.187	**12**	**0.001**	3
r_3	**1**	**0.077**	12	0.187	1
r_4	**59**	2.219	62	**0.062**	14
r_5	**38**	**1.421**	92	2.202	8

in $Time$ than in $|W| = 10,000$. We can prove again experimentally that WSPR and WSPR w/o heuristics are polynomial algorithms proportional to $|W|$.

From Table 7.26 to Table 7.32, we list the comparison results of WSPR and WSPR w/o heuristics of the nine test sets in $erTS$ domain. The results of the five test requests of $erTS$ with $|W| = 1,000$ are shown in Table 7.26. Both algorithms do not have trouble in computing the solutions. However, WSPR overruns WSPR w/o heuristics. Regarding $Time$, both are nearly the same.

As shown in Table 7.27, both algorithms have no problem in calculating solutions, and WSPR overruns WSPR w/o heuristics. The $Time$ of WSPR is slightly faster than WSPR w/o heuristics. From less $Time$ and smaller $\#W$, we can infer

Table 7.27: Results of $erTS$ with $|W| = 3,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**56**	**5.703**	59	6.296	11
r_2	**56**	**4.858**	89	5.405	8
r_3	**19**	**4.766**	24	6.233	9
r_4	**22**	**4.672**	25	6.344	8
r_5	**26**	**4.625**	36	5.891	8

Table 7.28: Results of $erTS$ with $|W| = 5,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**19**	13.719	46	**11.14**	7
r_2	**13**	**9.094**	23	10.516	7
r_3	**15**	**8.202**	40	10.609	8
r_4	**22**	**8.984**	42	11.263	9
r_5	**12**	**8.625**	24	10.202	7

Table 7.29: Results of $erTS$ with $|W| = 10,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**12**	12.532	33	**11.922**	6
r_2	**15**	11.532	30	**10.031**	6
r_3	**8**	12.53	18	**11.437**	6
r_4	**14**	11.358	34	**11.312**	7
r_5	**15**	**10.812**	35	11.062	6

that the heuristics of WSPR rather saves the computation time by reducing $\#W$.

Table 7.28 reveals that both algorithms have no trouble in computing the solutions, and WSPR overruns WSPR w/o heuristics. As a whole, $\#W$ of WSPR is twice less than that of WSPR w/o heuristics, meaning that the heuristics of WSPR is in effect. Regarding $Time$, both are similar.

The results of the five test requests of $erTS$ with $|W| = 10,000$ are displayed in Table 7.29. Like in previous cases, both algorithms have no difficulty in finding solutions, and WSPR overruns WSPR w/o heuristics in terms of $\#W$.

As shown in Table 7.30, both algorithms do not have trouble in computing solutions when $|W| = 20,000$. Like in previous results, in terms of $\#W$, WSPR

Table 7.30: Results of $erTS$ with $|W| = 20,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**15**	**36.859**	36	38.094	9
r_2	**10**	**21.125**	31	21.469	5
r_3	**19**	**32.516**	37	33.577	8
r_4	**15**	**29.297**	33	33.25	8
r_5	**10**	**30.172**	25	31.375	7

Table 7.31: Results of $erTS$ with $|W| = 30,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**14**	**47.985**	71	71.062	9
r_2	**12**	48.75	30	**47.985**	9
r_3	**20**	51.344	56	**50**	10
r_4	**14**	55.437	35	**42.812**	7
r_5	**14**	**45.687**	54	45.921	9

overruns WSPR w/o heuristics by using just half the number of web services required by WSPR w/o heuristics. Regarding $Time$, WSPR is slightly faster.

Table 7.31 shows the results of the five test requests of $erTS$ with $|W| = 30,000$. The results are similar to previous cases in that both algorithms have no difficulty in computing solutions, and WSPR overruns WSPR w/o heuristics in terms of $\#W$. Regarding $Time$, both have similar results. Interestingly, as a whole, WSPR is approximately five times slower in $Time$ than in $|W| = 10,000$. Thus, it can be proved experimentally that WSPR and WSPR w/o heuristics are polynomial algorithms proportional roughly to $2 \times |W|$ in this domain.

The results of the five test requests of $erTS$ with $|W| = 50,000$ are shown in Table 7.32. Similar to the previous cases, WSPR overruns WSPR w/o heuristics in terms of $\#W$. In the second and third requests, WSPR finds optimal solutions. Interestingly, WSPR is roughly ten times slower in $Time$ as $|W|$ increase from $|W| = 10,000$ to $|W| = 50,000$. This result further supports that WSPR and WSPR w/o heuristics are polynomial algorithms proportional roughly to $2 \times |W|$.

Tables 7.33, 7.34 and 7.35 demonstrates how effectively WSPR and WSPR w/o heuristics can solve requests as the test size increases from 1,000 to 50,000. The

Table 7.32: Results of $erTS$ with $|W| = 50,000$

test requests	WSPR		WSPR w/o heuristics		Lower bound
	$\#W$	$Time$	$\#W$	$Time$	$\#W$
r_1	**8**	**87.032**	30	88.25	7
r_2	**7**	85.407	28	**82.733**	7
r_3	**8**	120.233	33	**116.108**	8
r_4	**11**	**103.984**	37	108.484	8
r_5	**11**	94.468	35	**92.769**	7

Table 7.33: Effectiveness of WSPR over the baTS test sets in terms of $\#W$

$\|W\|$ (1,000)	1	3	5	10	20	30	50
WSPR	**10.4**	**24.2**	**61**	**23**	**19.6**	**8.6**	**13.2**
WSPR w/o heuristics	10.6	30.8	92	49	69.2	41.6	48.8
Lower bound	2.8	5.2	7.4	4.8	4.8	4	4.6

Table 7.34: Effectiveness of WSPR over the nwsTS test sets in terms of $\#W$

$\|W\|$ (1,000)	1	3	5	10	20	30	50
WSPR	**22.8**	**20.2**	**30**	**19.8**	**9.6**	**12.8**	**11.2**
WSPR w/o heuristics	31.4	36.4	50.2	57	26.2	46.6	25.6
Lower bound	10.6	8.4	10.6	13	9.6	10.8	8.6

Table 7.35: Effectiveness of WSPR over the erTS test sets in terms of $\#W$

$\|W\|$ (1,000)	1	3	5	10	20	30	50
WSPR	**29.8**	**35.8**	**16.2**	**12.8**	**13.8**	**14.8**	**9**
WSPR w/o heuristics	32.4	46.6	35	30	32.4	49.2	32.6
Lower bound	6.8	8.8	7.6	6.2	7.4	8.8	7.4

$\#W$ of WSPR and WSPR w/o heuristics at each size is averaged over the solutions to the five requests. The lower bound of $\#W$ is also averaged over the five lower bounds for each of solutions. Regarding $\#W$, WSPR and WSPR w/o heuristics do not have trouble in addressing requests even when $|W| = 50,000$. However WSPR overruns WSPR w/o heuristics completely. Conclusively, WSPR is more effective than WSPR w/o heuristics.

Regarding $Time$, Tables 7.36, 7.37 and 7.38 show how efficiently WSPR and WSPR w/o heuristics can solve requests as the test size increases from 1,000 to 50,000 in each of $baTS$, $nwsTS$ and $erTS$. WSPR and WSPR w/o heuristics

Table 7.36: Scalability of WSPR over the baTS test sets in terms of $Time$

$\|W\|$ (1,000)	1	3	5	10	20	30	50
WSPR	**0.7**	11.2	**23.1**	**32.1**	**66.6**	**81.3**	**136.6**
WSPR w/o heuristics	0.8	**6.7**	23.7	34.5	67.0	89.0	164.8

Table 7.37: Scalability of WSPR over the nwsTS test sets in terms of $Time$

$\|W\|$ (1,000)	1	3	5	10	20	30	50
WSPR	1.25	**7.34**	7.6	17.62	24.62	40.5	68.64
WSPR w/o heuristics	**1.19**	7.45	**7.19**	**16.97**	**24.03**	**39.29**	**67.09**

Table 7.38: Scalability of WSPR over the erTS test sets in terms of $Time$

$\|W\|$ (1,000)	1	3	5	10	20	30	50
WSPR	1.0746	**4.9**	**9.7**	11.7	**29.9**	**49.8**	98.2
WSPR w/o heuristics	**0.7**	6.0	10.7	**11.1**	31.5	51.5	**97.6**

scale up smoothly. Thus, WSPR and WSPR w/o heuristics are efficient and we can conclude that the two step search approach is in effect for increasing the computational efficiency.

Figure 7.1 shows how WSPR and WSPR w/o heuristics operate in the $baTS$ domain as the test size increases from 1,000 to 50,000. Note that the $\#W$ and $Time$ at each size are averaged over the solutions to the five requests. The line with triangle-mark at the bottom in the left graph of Figure 7.1 links values where each value is averaged over the five lower bounds of optimal solutions at each size. Regarding $\#W$, WSPR and WSPR w/o heuristics have no difficulties to address requests even when $|W| = 50,000$. However, WSPR overruns WSPR w/o heuristics completely. Regarding $Time$, WSPR and WSPR w/o heuristics scale up smoothly, with WSPR being slightly faster than WSPR w/o heuristics. We can simply assert that WSPR will continue to perform with this increasing smooth pattern in terms of $Time$, even if $|W|$ continues to increase. Thus, the time to compute the heuristics in the regression searching step is trivial and WSPR is scalable when the underlying web-service network topology follows the scale-free network topology.

Figure 7.2 deals with the $nwsTS$ domain by showing how WSPR and WSPR w/o heuristics operate as the test size increases from 1,000 to 50,000. Regarding

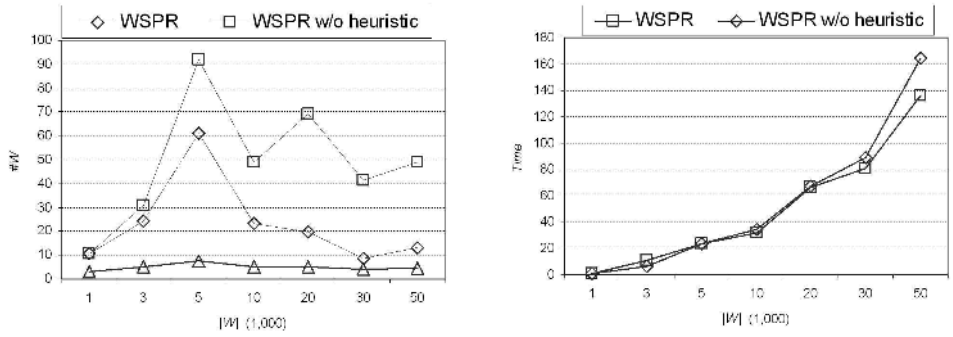


Figure 7.1: Scalability and effectiveness of WSPR over the baTS test sets. (left) $\#W$. (right) $Time$

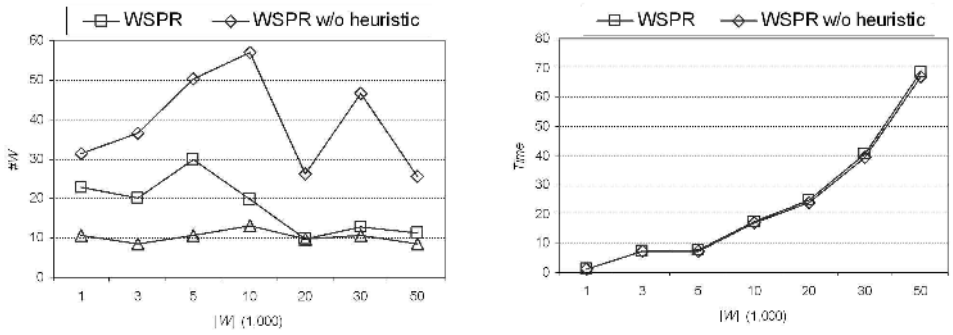


Figure 7.2: Scalability and effectiveness of WSPR over the nwsTS test sets. (left) $\#W$. (right) $Time$

$\#W$, as in $baTS$ domain, WSPR and WSPR w/o heuristics have no trouble in addressing requests, even when $|W| = 50,000$, but WSPR overruns WSPR w/o heuristics completely. Regarding $Time$, similar to the $baTS$ case, WSPR and WSPR w/o heuristics scale up smoothly without differences between them. In this domain, we can also simply assert that WSPR will continue to work with this increasing smooth pattern in terms of $Time$ even if $|W|$ continues to increase. Thus, WSPR is scalable when the underlying web-service network topology follows the small-world network.

Figure 7.3 deals with the $erTS$ domain by showing how effectively and efficiently WSPR and WSPR w/o heuristics can solve requests as the test size increases from 1,000 to 50,000. The results are similar to $baTS$ and $nwsTS$. That is, we can hold our assertion that WSPR will continue to work with this smooth

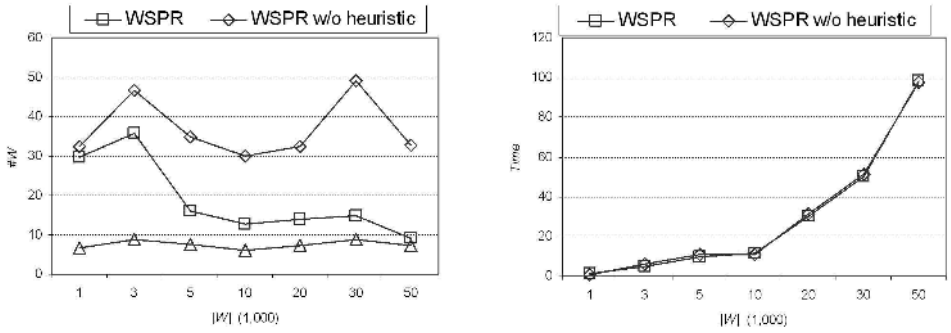


Figure 7.3: Scalability and effectiveness of WSPR over the erTS test sets. (left) $\#W$. (right) $Time$

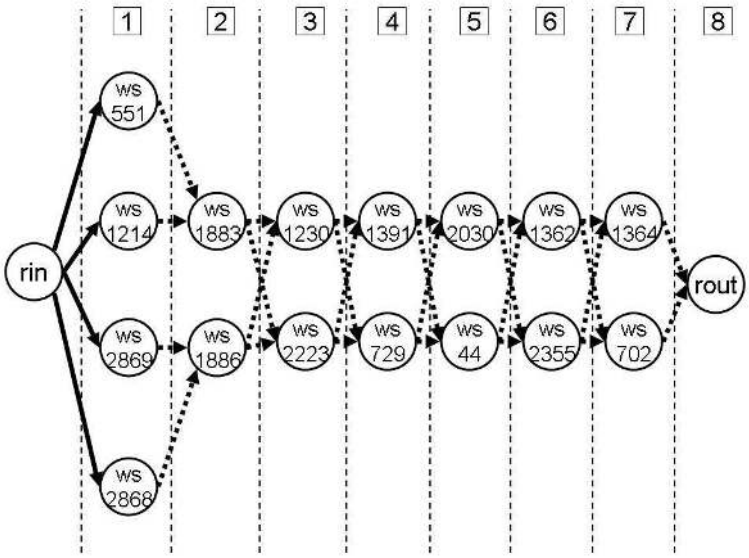


Figure 7.4: A solution of WSPR to $nwsTS$ with $|W| = 3,000$

increasing pattern in terms of $Time$, even if $|W|$ becomes large. As a result, WSPR remains scalable even in the case where the underlying web-service network topology follows the random network.

For a better understanding of the difference between WSPR and WSPR w/o heuristic in terms of $\#W$, we can conduct a statistical analysis on $\#W$ results data. Remember that we generate 21 test sets (seven test sets for each of $baTS$, $nwsTS$, and $erTS$). An paired T-test analysis[7] showed significant differences between competitors ($p = 0.008 < 0.05$). The low p value indicates a strong probability

[7]This analysis was conducted using Data Analysis ToolPak offered by MS-Office Excel

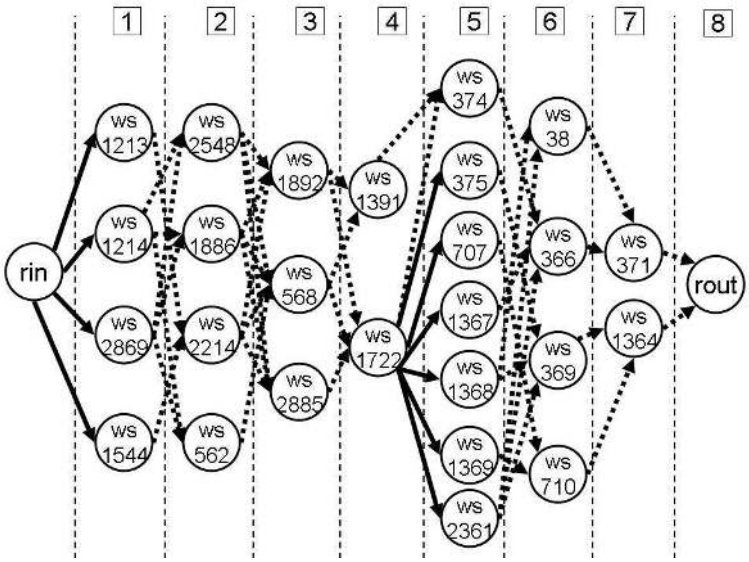


Figure 7.5: A solution of Blackbox to $nwsTS$ with $|W| = 3,000$

that one method's mean rank is significantly different from another, because p value means the probability that WSPR and WSPR w/o heuristic are samples from populations with the same mean. From this analysis, we can conclude that the WSPR's heuristic is in effect and its underlying hypothesis is is true. The hypothesis in Chapter 6 was as follows:

Hypothesis-1: *Choosing a web service with a greater contribution to match subGoal earlier in the regression search helps reach the initial state faster.*

7.3.5 WSPR and Blackbox

Figures 7.4 and 7.5 illustrate the solutions of WSPR and Blackbox to the fourth request of $nwsTS$ with $|W| = 3,000$, respectively. Note that WSPR and Blackbox have the same scheme in their forward searching stage because both aim to minimize the number of time steps to reach the goal. Therefore, as shown in the figure, both WSPR and Blackbox have the same number of time steps of eight. However, in their regression searching schemes, considerable differences exist. As mentioned in Section 3.3, when the size of a test set becomes very large, Blackbox skips performing a form of IDA* search. Instead, Blackbox converts the plan graph into a set of clauses that form a satisfiability problem (SAT problem). For the SAT prob-

lem, Blackbox applies Walksat, which belongs to the incomplete local SAT search algorithms. The algorithm[8] is limited to seek an assignment of the variables that satisfies a given formula without considering full- or partial matching. It is natural because the SAT problem is a decision problem, not an optimization problem. As a result, Walksat search strategy can produce poor solutions in the WSC problem domain.

For example, at the sixth time step in Figure 7.5, Blackbox composes a set of four web services, such as {`ws38`, `ws366`, `ws369`, `ws710`}, to match the sub-goal at the seventh time step that is the set of input parameters of two web-services, such as {`ws371`, `ws1364`}. This decision results in an exponentially increasing number of web services for the subsequent sub-goal. Due to the exponentially growing number, as previous results show, the regression search of Blackbox often fails before it reaches the initial state. On the contrary, at the same time step, WSPR, which is designed to favor the full-matching web services, composes a set of two web services, such as {`ws1362`, `ws2355`}, to match the sub-goal that the set of input parameters of two web services, such as {`ws1364`, `ws702`}. Thus, the size of subsequent sub-goal does not explode and continues this pattern until it reaches the initial state, thereby reducing $\#W$.

7.4 Summary

The experiments above, based on three human or machine-generated test sets, show that WSPR is capable of solving the problems better than other off-the-shelf AI planners in terms of efficiency, effectiveness, and scalability. In particular, WSPR tends to have a better performance than other AI planners when the underlying web service network topology is the scale-free network. Indeed, the other AI planners do not operate at all when the underlying web service network follows the scale-free network and the test size exceeds 5,000. Note that the size of the parameter set is the largest in the scale-free network case, and the computational complexity of the WSC problem exponentially increases as the size of the parameter set increases. This implies that the other AI planners are not as scalable as

[8]See Section 3.3 for more details about two search strategies of Walksat: a random walk strategy and a greedy strategy

WSPR.

WSPR also demonstrate a better performance over WSPR w/o heuristics, regardless of the different topologies of underlying web-service networks and the test set sizes. Thus, the strategy in support of locating fully-matching web service first in a tie situation is in effect. The experiments also proved that the two stage approach of WSPR has a significant effect on reducing computational time and establishing scalability because $Time$ of both WSPR and WSPR w/o heuristics did not blow up exponentially when the test set size increases. Finally, the experiments showed the usefulness of the suggested test set framework because it helps explain how algorithms run in different environments and test scenarios (e.g., small versus large sizes; diverse parameter cluster network topologies such as scale-free, small-world and random networks; and short versus long solution length). We recognize that the AI planning community can view WSPR as a customized algorithm for the WSC problem. As confirmed by the experimental evaluation, however, WSPR can be seen as a robust solution for the WSC problem because it can perform well in diverse WSC scenarios, as well as in the large number of web services.

The main findings about WSPR from these experimental results are as follows:

1. Effectiveness: it is defined as the ability to achieve stated goals or objectives, judged in terms of both output and impact. Our objective was to compose web services to generate the desired service with a minimum of web services. As confirmed by the experimental evaluation, WSPR shows better results in 80% of all the cases in terms of $\#W$, compared to the other prominent AI planners including Blackbox, IPP, and Graphplan. In particular, WSPR dominated WSPR w/o heuristics. This implies that the heuristics of WSPR with the strategy in support of locating fully-matching web service first in a tie situation, is in effect.

2. Computational efficiency: we measured how quickly WSPR generates the correct solutions in comparison with the other AI planners, as more data are applied to the problem. The experimental validation showed that WSPR outperformed the other competitors with significant differences in terms of $Time$.

3. Scalability: an algorithm is scalable if it can adapt to increased workloads and

continues to function well. The experiments proved that WSPR and WSPR w/o heuristics did not blow up exponentially when the test set size increases. This implies that both algorithms are scalable and the two step approach of WSPR is responsible for scalability. Note that WSPR and WSPR w/o heuristics perform the two step search process but WSPR w/o heuristics does not utilize the heuristic in the regression search step.

4. Robustness: an algorithm is considered to be robust if its' performance remains relatively stable, with a minimum of variation, even though its application domain changes. As confirmed by the experimental evaluation, WSPR can be seen as a robust solution to the WSC problem, because it can perform persistently well in diverse WSC scenarios that arise in different test sets, including EEE05, ICEBE05 characterized by the tree structure, and WSBen generated test sets: baTS, newTS, and erTS featured by complex and random graph structures.

Chapter 8

Application of Semantic Web-service Composition in Manufacturing

The process to obtain interoperability between manufacturing and logistic companies takes a long time because companies have heterogeneous information systems that are not designed for interoperability. Semantic web-service technologies have the potential of saving the time needed for obtaining the interoperability by automating these tasks. In this chapter, we propose an ontology-based framework using semantic web-service technologies to secure the reliable and large scale interoperability among the distributed entities of a manufacturing systems, namely: a design firm, manufactures, and third party logistics providers. We review existing semantic web-service technologies and propose an approximate scenario that forms the motivating base for our proposed framework. Currently, this work is only at the conceptual stage, with a focus on proposing a software design rather than implementing the proposed framework.

8.1 Research Background

In the past, most companies were able to reduce manufacturing costs and maintain consistent quality by mass production because demands were stable, markets were homogeneous, and product life cycles were long [43]. More recently, markets have been changed into volatile environments characterized by mass customization. Typically, production life cycle is shortened, marketing powers are shifted

towards buyers who require individual customization, and markets become highly diversified and global [56]. The use of a loosely integrated virtual enterprise based framework holds the potential of surviving the changing market needs [61]. As a result, the growth on both the volume and scope in the e-business for the manufacturing domain is fast.

However, since manufacturing and logistic companies use heterogeneous information systems, business to business (B2B) integration is necessary for conducting the e-business [72]. Automated transaction between companies can save time and be less error-prone than having people process repetitive information [60]. The use of standards (e.g., RosettaNet[1]) is considered to be an extendable solution for many partners to enable such automatic transactions between them. In order to use the standards, partners must extract the internal contents for automated transactions from their information systems. This is because these standards provide high flexibility at the implementation level, such as message contents and how processes are composed. Therefore, partners have to make a significant effort to implement exactly how a standard is used, even though they agree on the standard. Consequently, the standard-based B2B integration between manufacturing companies can take a significant amount of time and effort.

Web services have added a new level of functionality to the current web by taking a first step towards seamless integration of distributed software components using web standards. Nevertheless, current Web-service technologies around SOAP, WSDL, and UDDI operate at a syntactic level, although they support interoperability between the many diverse application development platforms that exist today through common standards. They, however still require human interaction to a large extent. For example, the human programmer has to manually search for appropriate web services in order to combine them in a useful manner, which limits scalability and greatly curtails the added economic value envisioned with the advent of web services.

Semantic web services (SWS) introduce new technologies for information system integrations which can significantly speed up the integration process, thus enabling seamless interoperation between different information systems while keeping human intervention to a minimum. In other words, if business partners describe

[1]http://www.rosettanet.org/Rosettanet/Public/PublicHomePage

their service properties, capabilities, and interfaces (how to interact with them) in computer understandable form [61] using SWS technologies, it is possible to use mediation technology to adapt the interaction of another business partner to be compatible [89]. Although several initiatives, such as OWL-S [95, 110, 109], WSMO[2], and WSDL-S[3], have emerged in this area aiming at addressing the problem of semantics in web services, we will mainly use OWL-S to address our problem of achieving B2B integration between manufacturing and logistic companies who use heterogeneous information systems.

8.2 Motivating Scenario

Consider a scenario in a design firm, where there is an order requesting the manufacturing of gear by a cutting process out of forged raw material. The forging manufacturing process has two sub-types: conventional and precision forging processes [105]. The conventional forging process leads to raw parts with rough surfaces and large allowances, which is the amount of material to be removed in the final manufacturing steps. On the other hand, in the case of the precision forging process, raw parts are characterized by small allowances and are hardened with semi-rough surfaces. In addition to the order of the manufacturing of a gear, the design firm makes another order to request the delivery of the produced gears from the manufacturer. The design firm prefers both the manufacturing and logistics company to be co-located in Pennsylvania, where the design firm is located.

If the design firm has no manufacturers and logistics providers with prior relationships, the design firm may have to use an ad hoc, time-consuming, and error-prone process to locate the right business partners. However, if each business partner provides its service using semantic web services encoded in computer understandable form, then we can describe an automatic information workflow between business partners using the UML activity diagram as shown in Figure 8.1. The activity diagram has four basic collaboration steps: (1) Service specification, (2) Matchmaking, (3) Negotiation and bidding, and (4) Contracting.

In order to seamlessly integrate the basic collaboration steps, we propose a

[2]http://www.wsmo.org
[3]http://www.w3.org/Submission/WSDL-S/

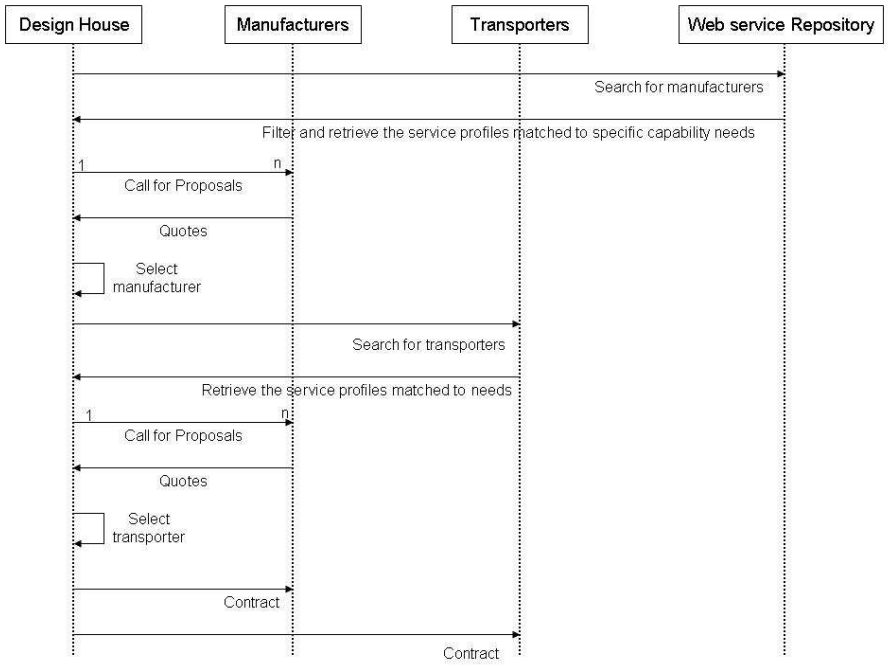


Figure 8.1: Activity diagram for the motivating example

four-step based approach in Section 8.4 using the semantic web service technology.

8.3 Semantic Web Services

Web services require the description of each service so that other services can understand its features and learn how to interact with it. The representative language for describing operational features of web services is WSDL (Web Services Description Language) [115]. WSDL is being standardized within the World Wide Web Consortium(W3C). Major industry leaders are supporting and participating in WSDL development. Therefore, WSDL is likely to gain considerable momentum as the language for web service description. However, WSDL provides little or no support for semantic description of web services. It mainly includes constructs that describe web services from a syntactic point of view. To implement semantic web-enabled web services, it is required to extend WSDL with semantic capabilities. This is also known as "lifting". This extension would lay the groundwork for the automatic selection and composition of web services. A web ontology languages,

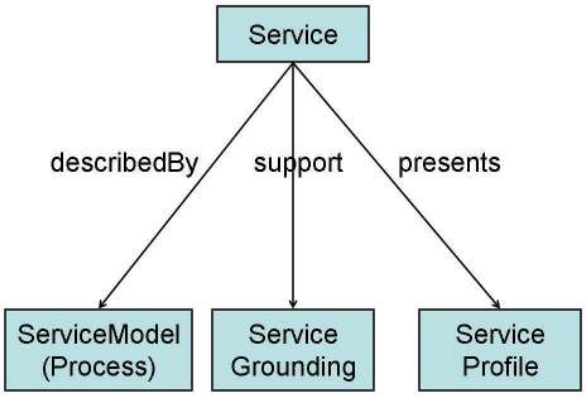


Figure 8.2: Top-level OWL-S classes and their relationships

such as OWL [109] may be used to specify the proposed semantics. It is an object-oriented language describing semantics in terms of classes, properties, and axioms (e.g., subsumption relationships between classes or properties). OWL builds on earlier web ontology standards, such as RDF and RDF Schema, and extends those languages with richer modeling primitives (e.g., cardinality).

OWL-S is an ontology of service concepts that supplies a web service designer with a core set of markup language constructs for describing the properties and capabilities of a web service in an unambiguous, computer-interpretable form [110, 95]. Following the layered approach to markup language development, the current version of OWL-S builds on OWL by the W3C. OWL-S introduces ontologies to describe the concepts in the services' domain (e.g., flights and hotels, tourism, e-business), and generic concepts to describe the services themselves (e.g., control flow, data flow) and how they relate to the domain ontologies (via inputs and outputs, preconditions and effects, and so on). These semantically rich descriptions enable automated machine reasoning over service and domain descriptions, thus supporting automation of service discovery, composition, and execution, while reducing manual configuration and programming efforts.

OWL-S organizes a service description into four conceptual areas, as shown in Figure 8.2: (1) Process model, (2) Profile, (3) Grounding, and (4) Service.

- Process model: A process model describes how a service performs its tasks. It includes information about inputs, outputs (including a specification of the

conditions under which various outputs will occur), preconditions (circumstances that must hold before a service can be used), and results (changes brought about by a service). The process model differentiates between composite, atomic, and simple processes. For a composite process, the process model shows how it breaks down into simpler component processes, and it displays the flow of control and data between them. Atomic processes are essentially "black boxes" of functionality, and simple processes are abstract process descriptions that can relate to other composite or atomic processes.

- Profile: A profile provides a general description of a web service, and it is intended to be published and shared to facilitate service discovery. Profiles can include both functional properties (inputs, outputs, preconditions, and results) and nonfunctional properties (service name, text description, contact information, service category, and additional service parameters). The functional properties are derived from the process model, but it is not necessary to include all the functional properties from the process model in a profile. A simplified view can be provided for service discovery, based on the assumption that the service consumer would eventually look at the process model to achieve a full understanding of how the service works.

- Grounding: A grounding specifies how a service is invoked by detailing how the atomic processes in a services process model map onto a concrete messaging protocol. OWL-S provides for different types of grounding to be used, but the only type developed to date is the WSDL grounding, which allows any web service to be marked as an semantic web services using OWL-S.

- Service: A service simply binds the other parts together into a unit that can be published and invoked. It is important to understand that the different parts of a service can be reused and connected in various ways. For example, a service provider may connect its process model with several profiles in order to provide customized advertisements to different communities of service consumers. A different service provider that offers a similar service may reuse the same process model, possibly as part of a larger composite process, and connect it to a different grounding. The relationships between service components are modeled using properties, such as presents

(Service-to-Profile), described By (Service-to-Process Model), and supports (Service-to-Grounding) as shown in Figure 8.2.

Note that process models of OWL-S employs the orchestration-based service composition approach, where one executable business process may interact with both internal and external web services. In other words, the composed service operates as the hub station for data-flow and control-flow. Differing from the centralized orchestration approach, the choreography-based service composition approach is more collaborative in nature, where each party involved in the process describes the partial role in the interaction. Consequently, there is no one party that virtually owns the conversation initiative. In the industry standards, Web Service Choreography Interface (WSCI)[4] is designed for implementing the choreography-based service composition, while the Business Process Execution Language for Web Services (BPEL4WS)[5] intends to realize the orchestration-based service composition. Liu et al. [66] proposed a protocol called Flow-based Infrastructure for Composing Autonomous Services (FICAS), which supports the choreography based service composition paradigm. They showed that FICAS can reduce the amount of data traffic significantly by moving computations closer to the data.

8.4 Overview of the Proposed Framework

Based on the motivating example in Section 8.2, we propose an approach for the automatic composition of services provided by distributed business partners, as shown in Figure 8.3. This approach consists of four conceptually separate phases: service specification, matchmaking, negotiation and bidding, and generating composite services. This approach is essentially in accordance with the framework proposed by Medjahed et al. [72]. However, our proposal adopts negotiation and bidding step to select the best business partner while Medjahed et al. use a simple selection algorithm based on the QoS (quality of service) parameters specified in the web-service profiles. We believe that our approach can generate more feasible solutions than the previous approach because it can induce business partners' in-

[4]http://www.w3.org/TR/wsci/

[5]http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

Table 8.1: OWL primitives in DL terms

DL Syntax	OWL Syntax	Serv. Descript. Lang.
A	owl:Class	Concept
$\top$	owl:Thing	Thing
$\bot$	owl:Nothing	Nothing
$(C \subseteq D)$	owl:subClassOf	Subsumption
$(C \equiv D)$	owl:sameClassAs	Equivalence
R	owl:Property	Properties
R	owl:ObjectProperty	ObjectProperties
$(C \sqcap D)$	owl:intersectionOf	Conjunction
$(C \sqcup D)$	owl:disjunctionOf	Disjunction
$(\neg C)$	owl:complementOf	Negation
$(\forall R.C)$	owl:toClass	Universal Role Rest.
$(\exists R.C)$	owl:hasClass	Existential Role Rest.
R^{-}	owl:inverseOf	Inverse Roles
$(R \subseteq S)$	owl:subPropertyOf	Subsumption of Roles
$(R \equiv S)$	owl:samePropertyAs	Equivalence of Roles
$\{o\}$	XML Type + rdf:value	Nominals
$\exists T.\{o\}$	owl:hasValue	Value Restrictions

volvement and competition in terms of prices as well as quality. However, it may pose problems related to convergence.

- Specification phase: The specification phase enables high-level descriptions of the desired service. For the description purpose, we can use Description Logics (DL) language. DL is used to express the subsumption, equality, and set definitions between classes and objects. There are both major DL languages such as DAML-DL (DAPPA Markup Language) [108] and OWL-DL, which are compatible because OWL-DL is based on DAML-DL. DL requires a reasoning service. The Java-based Expert System (JESS) [42] implements the interface APIs for DAML-DL so that JESS can translate DAML-DL into a set of predicates (ordered-facts) consisting of property or verb, subject, and object. Suppose that partners' service profiles are translated into the JESS knowledge base. Then, we can filter desired service profiles via DAML-DL as a query language. Likewise, there are OWL-DL reasoners, such as RACER [49] and Pellet [85], which will be useful.

  Another approach to describe the desired service is to use Composite Service

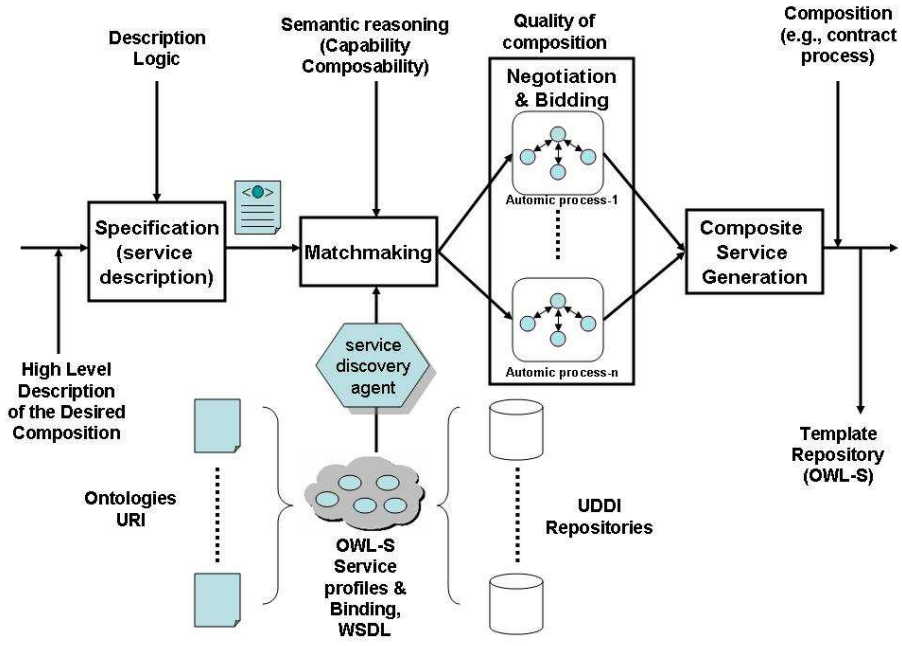


Figure 8.3: Overview of the proposed approach for service composition

Specification Language (CSSL) [72]. CSSL aims at providing the high-level descriptions of composite services. Composers only need to have a general idea about the service they are interested in offering (e.g., the operations to be outsourced). They are not required to be aware of the full technical details, such as descriptions of the component services, their characteristics (e.g., data types), and how they are plugged together.

CSSL does not have a powerful reasoning engine like JESS, RACER, or Pallet. Meanwhile JESS, RACER and Pallet are not appropriate in themselves for composing multiple service profiles. For these reasons, CSSL and DL can be considered compatible with each other. In other words, the high-level composite service requirement can be described in CSSL, and the detailed atomic services of the composite service can be described in DL.

- Matchmaking phase: Once specifications are provided, the next step is to filter service profiles that conform to the specifications defined in the previous step. If the requirements are described in OWL-DL, RACER or Pellet can be exploited to filter the service profiles.

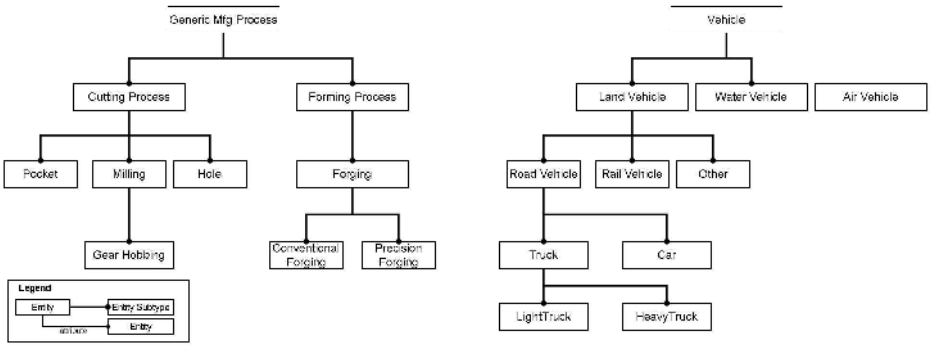


Figure 8.4: Partial ontologies in EXPRESS-G format. (left) Manufacturing process. (right) Vehicle

- Negotiation and bidding phase: In the previous matching phase, service requesters have the possibility of finding a large number of service profiles conforming to the requirements. In this case, it is a crucial issue to locate a partner who can provide a service (e.g., manufacturing or logistics) with the minimum cost or the highest quality. To search for the best partner, the service requester can send bid requests to the companies filtered in the previous matchmaking phase, and choose the partner who offers the lowest cost with high quality.

- Composite service generation: The last phase aims at generating a detailed description of a composite service. This includes the list of outsourced services, mappings between composite service and component service operations, mappings between messages and parameters, and flow of control and data between component services. We can use an OWL-S process model to describe how the desired composite service can be broken down into simpler concrete services provided by business partners, as well as the flow of control and date between the services.

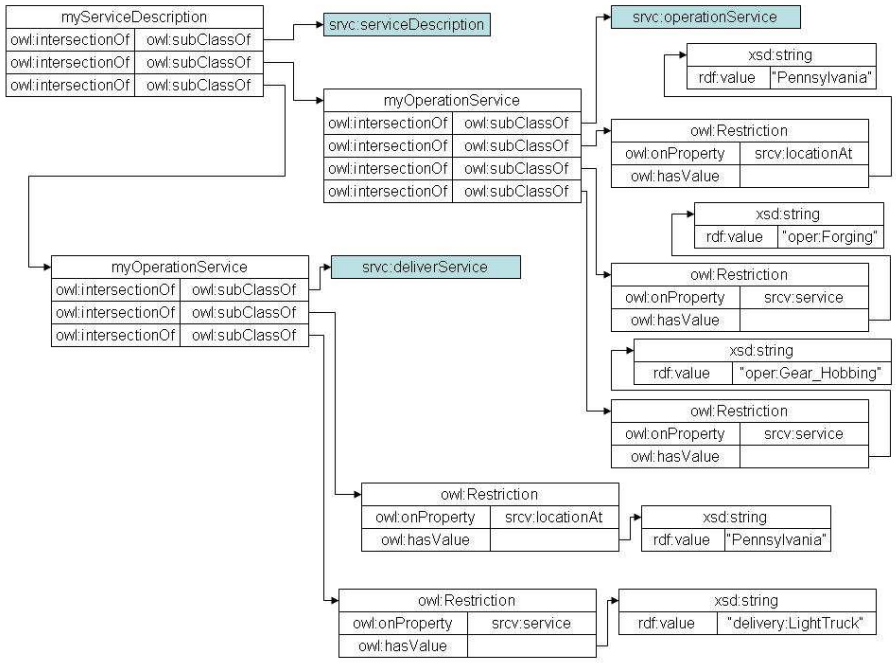


Figure 8.5: The service description of the motivating scenario

8.5 Illustrative Example

8.5.1 Specification phase

We now describe the service request of the design firm in the motivating scenario in terms of OWL-DL. The primitives of OWL-DL is explained in Table 8.1. When services are discovered and selected, the business level description of the services plays a key role. In our problem domain, this will be a description of what manufacturing process are to be served, which vehicle is being used, and where they must be located. At the service selection stage, we can describe the requirement of the design firm as follows:

- $S_{myService} \equiv (srvc : operationService \sqcap \exists srvc : locationAt.\{Pennsylvania\} \sqcap \exists srvc : service.\{oper : Forging\} \sqcap \exists srvc : service.\{oper : GearHobbing\}) \sqcap (srvc : VehicleService \sqcap \exists srvc : locationAt.\{Pennsylvania\} \sqcap \exists srvc : service.\{oper : LightTruck\})$

This service description states that the design firm requires: (1)A manufacturing company which is located in Pennsylvania and has the capability to provide

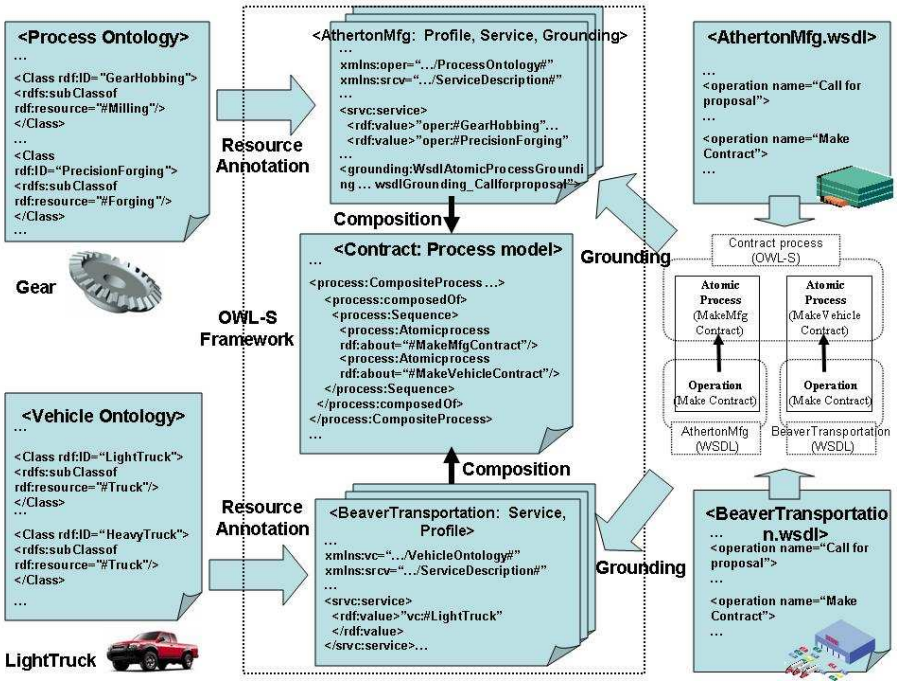


Figure 8.6: Combination into the OWL-S framework

manufacturing operations, such as forging and gear hobbing; and (2) A logistic company which can provide the delivery service with a light duty truck in Pennsylvania. The terms used in this description are defined in a service description ontology, a manufacturing process ontology, and a vehicle ontology. Figure 8.4 shows the taxonomy hierarchy of the manufacturing process and vehicle in terms of the EXPRESS-G format[6]. Based on the EXPRESS-G expression, we define the manufacturing process ontology and the vehicle ontology in OWL in Figures 8.7 and 8.8, respectively. This OWL-DL based service description has a corresponding class diagram that is shown in Figure 8.5.

We can also describe other service requirements using OWL-DL and the class diagram, such as what goods are to be transported, where they will be transported from, when the vehicle will depart, what its destination is, when it is expected to arrive, and other relevant terms of service, including insurance liability, cost and payment conditions.

[6]http://en.wikipedia.org/wiki/ISO_10303-11

8.5.2 Matchmaking and Negotiation phase

In this step, the design firm discovers the manufacturers and logistics providers that match the necessary service categories (e.g., manufacturing service, logistic service) from a web service registry (e.g., UDDI), and retrieves the service profiles of matched partners.

Concurrently, such a concrete and detailed service description of a service is not appropriate for service advertising and discovery because service requests and advertisements would have to include many such classes covering all acceptable service parameter configurations (e.g., detailed service cost per item) [89]. Instead, service requestors and providers are abstract from concrete parameter information, switching to less specific class descriptions. In such abstract service descriptions, they specify the set of concrete services that they are willing to accept. For example, we may have two companies, `AthertonMfg` and `BeaverTransportation`, which may advertise their services using the OWL-DL for abstract service descriptions as follows:

- $S_{AthertonMfg} \equiv srvc : operationService \sqcap \exists srvc : locationAt.\{Pennsylvania\} \sqcap \exists srvc : service.\{oper : PrecisionForging\} \sqcap \exists srvc : service.\{oper : GearHobbing\}$

This states that `AthertonMfg` can offer manufacturing services including the precision forging and gear hobbing operations, and is located in Pennsylvania. The OWL-S based service capability description which corresponds to this OWL-DL based statement are shown in Figure 8.9. `AthertonMfg` can register the OWL-DL based statement directly or the OWL-S file indirectly with the service discovery agent.

- $S_{BeaverTranportation} \equiv srvc : VehicleService \sqcap \exists srvc : locationAt.\{Pennsylvania\} \sqcap \exists srvc : service.\{oper : LightTruck\}$

This states that `BeaverTranportation` can offer a deliver service using a light duty truck carrier in Pennsylvania. The OWL-S based service capability description which corresponds to this OWL-DL based statement are shown in Figure 8.10. Similar to `AthertonMfg`, `AthertonMfg` can register the OWL-DL based statement directly or the OWL-S file indirectly with the service discovery agent.

When the service discovery agent receives a service request, it finds the set of advertisements which intersects with the request and returns the set. Note that an advertisements and a request intersect if they specify at least one common concrete service. The discovery agent can use DL reasoners, such as RACER or DAML-DL internally. In our motivating scenario, `AthertonMfg` can be discovered by the DL reasoners because the precision forging has the subsumption relation with the forging operation, which is the design house request. In addition, `AthertonMfg` can satisfy the other requests, such as the location in Pennsylvania and other manufacturing services of gear hobbing. Full details of the inferencing mechanism is not explained here. One good reference for the inferencing mechanism can be found in [49].

The list of services returned by the service discovery agent includes URIs referencing the service providers, allowing the requestor to make direct contact through the service providers' web services. In our example, the design firm makes contact with manufacturing and logistic service providers and selects the best ones. In order to do the selection, the design firm can activate the bidding process according to the protocols described in Figure 8.1.

For example, as shown in Figure C.15, `AthertonMfg` may provide an operation named `Call_for_Proposal` that receives the call for proposal from the service requestor and returns a proposal. In our motivating scenario, the call for proposal of the design firm may contain the number of gear parts that need to be manufactured and the expected cost of manufacturing them. Similarly, `BeaverTranportation` can offer an operation to process the call for proposal from the service requestor and returns a proposal. In our scenario, the call for proposal of the design firm to `BeaverTranportation` may contain when the light duty truck carrier will be used and how much the using cost will be.

8.5.3 Composite service generation phase

The last phase aims at generating a detailed description of a composite service. At this point, suppose that the design firm of the motivating scenario have chosen `AthertonMfg` and `BeaverTranportation` for its manufacturing and delivery outsourcing respectively, by establishing the bidding process. Figure 8.12 shows a com-

posite process to be used for making contracts with both outsourced companies. To create an instance of a composite process in OWL-S, *process:CompositeProcess* construct is used. In Figure 8.12, `Contract_Process` is the instance of a composite process and has a *process:sequence* primitive. By definition, a *sequence* in OWL-S is a sequence that performs other atomic or composite processes. Note that the atomic process named `MakeMfgContract` is grounded into the `MakeContract`, which is the operation of `AthertonMfg` web service as shown in Figure 8.6. Similarly, `MakeVehicleContract` is grounded into the `MakeContract` which is the operation of `BeaverTransportation` web service.

Regarding the execution of the OWL-S composite model, we can build an in-house application by using OWL-S API[7]. OWL-S API supports the execution engine for OWL-S process models. The OWL-S execution engine is used to invoke Web Services for gathering information that is used in the planning process. To our knowledge, it is the only open-source API designed specifically for working with OWL-S ontologies.

8.6 Future Work

The framework presented in this chapter is a suggestion and not a deployed application. For this reason, certain simplifications have been made which need to be revisited for future work. First, we need to employ AI planning techniques to automate the composition of web services. AI Planners use the description of the preconditions and effects of a service to do various sorts of reasoning about how to combine services into a composite service. OWL-S supports the description of the preconditions and effects of services using OWL statements. Therefore, AI planners need to have the capability to understand the semantics of OWL in order to evaluate such preconditions. However, typical AI planners support only syntactic-based reasoning or fairly limited semantic-based reasoning capability. As a result, it is important to augment AI planners with the OWL-S reasoning capability to overcome these problems. In particular, we are considering the extension of WSPR by adding the reasoning service, which can be carried out by combining existing DL reasoners, such as RACER-DL, or developing an in-house reasoner

[7]http://www.mindswap.org/2004/owl-s/api

using JENA[8] rules engine.

We also identify another challenge of writing the service descriptions effectively because the way of describing services for providers or requesters can affect the performance of reasoning significantly. For this purpose, we will investigate the current projects such as Athena and IV& I. Athena (Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Application) is an integrated project sponsored by the European Commission and NIST (The US National Institute of Standards and Technology). Its objective is to be the most comprehensive and systematic research initiative in the field of enterprise application interoperability, removing barriers to the exchange of information in and among organizations. It combines the latest research results in the three key areas of the interoperability problem space: (1)Business solutions (Enterprise modeling in the context of collaborative enterprises and cross-organizational business processes). (2)Knowledge representation and application solutions (Ontology-based systems and ontology modeling tools). (3) Modern ICT solutions (Web services, MDA approach, agent-based systems).

IV& I (Inventory Visibility and Interoperability) has sponsors including the Automotive Industry Action Group (AIAG), Original Equipment Supplier Association (OESA), and Odette, an industry association of European automotive manufacturers and suppliers. This project aims being able to seamlessly communicate demand throughout the supply chain. It could result in an estimated net savings to the auto industry of $255 million. The resulting programs will operate at the semantic level, not the data level, and exploit their approaches for the service description.

[8]http://www.hpl.hp.com/semweb/

```
<rdf:RDF
    xmlns:rdf=‘‘http://www.w3.org/1999/02/22-rdf-syntax-ns#’’
    xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema#’’
    xmlns:rdfs=‘‘http://www.w3.org/2000/01/rdf-schema#’’
    xmlns:owl=‘‘http://www.w3.org/2002/07/owl#’’
    xmlns=‘‘http://www.owl-ontologies.com/unnamed.owl#’’
  xml:base=‘‘http://www.owl-ontologies.com/unnamed.owl’’>
  <owl:Ontology rdf:about=‘‘’’/>
  <owl:Class rdf:ID=‘‘PrecisionForging’’>
    <rdfs:subClassOf>
      <owl:Class rdf:ID=‘‘Forging’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID=‘‘ConventionalForging’’>
    <rdfs:subClassOf>
      <owl:Class rdf:about=‘‘#Forging’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID=‘‘Hole’’>
    <rdfs:subClassOf>
      <owl:Class rdf:ID=‘‘Cutting’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID=‘‘Pocket’’>
    <rdfs:subClassOf>
      <owl:Class rdf:about=‘‘#Cutting’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID=‘‘Mfg_process’’/>
  <owl:Class rdf:ID=‘‘Milling’’>
    <rdfs:subClassOf>
      <owl:Class rdf:about=‘‘#Cutting’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about=‘‘#Cutting’’>
    <rdfs:subClassOf rdf:resource=‘‘#Mfg_process’’/>
  </owl:Class>
  <owl:Class rdf:ID=‘‘GearHobbing’’>
    <rdfs:subClassOf rdf:resource=‘‘#Milling’’/>
  </owl:Class>
  <owl:Class rdf:about=‘‘#Forging’’>
    <rdfs:subClassOf>
      <owl:Class rdf:ID=‘‘Forming’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about=‘‘#Forming’’>
    <rdfs:subClassOf rdf:resource=‘‘#Mfg_process’’/>
  </owl:Class>
</rdf:RDF>
```

Figure 8.7: Partial OWL encoding of the manufacturing process ontology

```
<rdf:RDF
    xmlns:rdf=‘‘http://www.w3.org/1999/02/22-rdf-syntax-ns#’’
    xmlns:xsd=‘‘http://www.w3.org/2001/XMLSchema#’’
    xmlns:rdfs=‘‘http://www.w3.org/2000/01/rdf-schema#’’
    xmlns:owl=‘‘http://www.w3.org/2002/07/owl#’’
    xmlns=‘‘http://www.owl-ontologies.com/unnamed.owl#’’
  xml:base=‘‘http://www.owl-ontologies.com/unnamed.owl’’>
  <owl:Ontology rdf:about=‘‘’’/>
  <owl:Class rdf:ID=‘‘RailVehicle’’>
    <rdfs:subClassOf>
      <owl:Class rdf:ID=‘‘LandVehicle’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID=‘‘WaterVehicle’’/>
  <owl:Class rdf:ID=‘‘RoadVehicle’’>
    <rdfs:subClassOf rdf:resource=‘‘#LandVehicle’’/>
  </owl:Class>
  <owl:Class rdf:ID=‘‘LightTruck’’>
    <rdfs:subClassOf>
      <owl:Class rdf:ID=‘‘Truck’’/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID=‘‘OtherVehicle’’>
    <rdfs:subClassOf rdf:resource=‘‘#LandVehicle’’/>
  </owl:Class>
  <owl:Class rdf:about=‘‘#Truck’’>
    <rdfs:subClassOf rdf:resource=‘‘#RoadVehicle’’/>
  </owl:Class>
  <owl:Class rdf:ID=‘‘Car’’>
    <rdfs:subClassOf rdf:resource=‘‘#RoadVehicle’’/>
  </owl:Class>
  <owl:Class rdf:ID=‘‘AirVehicle’’/>
  <owl:Class rdf:ID=‘‘HeavyTruck’’>
    <rdfs:subClassOf rdf:resource=‘‘#Truck’’/>
  </owl:Class>
</rdf:RDF>
```

Figure 8.8: Partial OWL encoding of the vehicle ontology

```
<rdf:RDF xmlns:rdf=‘‘http://www.w3.org/1999/02/22-rdf-syntax-ns#’’
xmlns:rdfs=‘‘http://www.w3.org/2000/01/rdf-schema#’’
xmlns:owl=‘‘http://www.w3.org/2002/07/owl#’’
xmlns:service=‘‘http://www.daml.org/services/owl-s/1.0/Service.owl#’’
xmlns:process=‘‘http://www.daml.org/services/owl-s/1.0/Process.owl#’’
xmlns:profile=‘‘http://www.daml.org/services/owl-s/1.0/Profile.owl#’’
xmlns:actor=‘‘http://www.daml.org/services/owl-s/1.0/ActorDefault.owl#’’
xmlns:oper=‘‘.../ProcessOntology.owl#’’
xmlns:srvc=‘‘.../ServiceDescription.owl#’’
xmlns=‘‘.../AthertonMfg.owl#’’> ... <profile:contactInformation>
     <actor:Actor rdf:ID=‘‘AthertonMfg’’>
     ...
     </actor:Actor>
</profile:contactInformation> ... <profile:ServiceProfile
rdf:ID=‘‘AtertonMfgServiceProfile’’>... <srvc:service>
    <rdf:value>‘‘oper:#GearHobbing’’</rdf:value>
    <rdf:value>‘‘oper:#PrecisionForging’’</rdf:value>
</srvc:hasProcess>... </profile:ServiceProfile>...

<grounding:WsdlAtomicProcessGrounding
rdf:ID=‘‘wsdlGrounding_CallforProposal’’>
<grounding:WsdlAtomicProcessGrounding
rdf:ID=‘‘wsdlGrounding_MakeContract’’>
<grounding:owlsProcess
rdf:resource=‘‘#MakeMfgContract’’>
...
</rdf:RDF>
```

Figure 8.9: Partial service profile of `AthertonMfg` encoded in OWL-S

```
<rdf:RDF xmlns:rdf=‘‘http://www.w3.org/1999/02/22-rdf-syntax-ns#’’
xmlns:rdfs=‘‘http://www.w3.org/2000/01/rdf-schema#’’
xmlns:owl=‘‘http://www.w3.org/2002/07/owl#’’
xmlns:service=‘‘http://www.daml.org/services/owl-s/1.0/Service.owl#’’
xmlns:process=‘‘http://www.daml.org/services/owl-s/1.0/Process.owl#’’
xmlns:profile=‘‘http://www.daml.org/services/owl-s/1.0/Profile.owl#’’
xmlns:actor=‘‘http://www.daml.org/services/owl-s/1.0/ActorDefault.owl#’’
xmlns:vc=‘‘.../VehicleOntology.owl#’’
xmlns:srvc=‘‘.../ServiceDescription.owl#’’
xmlns=‘‘.../BeaverTranportation.owl#’’> ...
<profile:contactInformation>
     <actor:Actor rdf:ID=‘‘AthertonMfg’’>
     ...
     </actor:Actor>
</profile:contactInformation>... <profile:ServiceProfile
rdf:ID=‘‘BeaverTransportation’’> ... <srvc:hasVehicle>
    <rdf:value>‘‘vc:#LightTruck’’</rdf:value>
</srvc:hasProcess> ... </profile:ServiceProfile> ...
<grounding:WsdlAtomicProcessGrounding
rdf:ID=‘‘wsdlGrounding_CallforProposal’’>
<grounding:WsdlAtomicProcessGrounding
rdf:ID=‘‘wsdlGrounding_MakeContract’’> <grounding:owlsProcess
rdf:resource=‘‘#MakeVehicleContract’’> ... </rdf:RDF>
```

Figure 8.10: Partial service profile of `BeaverTransportation` encoded in OWL-S

```
<?xml version=‘‘1.0’’?> <definitions name=‘‘AthertonMfg’’ ... > ...
<portType name=‘‘Interface_CFP’’>
    <operation name=‘‘Call_for_Proposal’’>
            <input message=‘‘MessageCFPIn’’/>
            <output message=‘‘MessageCFPOut’’/>
    </operation>
    <operation name=‘‘Make_Contract’’>
            <input message=‘‘MessageContractIn’’/>
            <output message=‘‘MessageContractOut’’/>
    </operation>
</portType> ... </definitions>
```

Figure 8.11: Partial WSDL of `AthertonMfg`

```
<rdf:RDF xmlns:rdf=‘‘http://www.w3.org/1999/02/22-rdf-syntax-ns#’’
xmlns:rdfs=‘‘http://www.w3.org/2000/01/rdf-schema#’’
xmlns:owl=‘‘http://www.w3.org/2002/07/owl#’’
xmlns:service=‘‘http://www.daml.org/services/owl-s/1.0/Service.owl#’’
xmlns:process=‘‘http://www.daml.org/services/owl-s/1.0/Process.owl#’’
xmlns:profile=‘‘http://www.daml.org/services/owl-s/1.0/Profile.owl#’’
xmlns:actor=‘‘http://www.daml.org/services/owl-s/1.0/ActorDefault.owl#’’
xmlns=‘‘.../AthertonMfg.owl#’’>
...
<process:CompositeProcess
rdf:ID=‘‘Contract_Process’’>
<process:composedOf>
<process:Sequence>
<process:components rdf:parseType=‘‘Collection’’>
    <process:AtomicProcess rdf:about=‘‘#MakeMfgContract’’ />
    <process:AtomicProcess rdf:about=‘‘#MakeVehicleContract’’ />
</process:components> </process:Sequence> </process:composedOf>
</process:CompositeProcess>
...
</rdf:RDF>
```

Figure 8.12: Partial OWL-S process model for contracting suppliers

Chapter 9

Conclusions and Future Research

The service oriented architecture becomes IT infrastructure intermediary, in which web services can be securely published, reused, and invoked as part of distributed applications. They also offer exceptional opportunities for collaboration and coordination to people across a wide range of fields including industry, academia, and government. However, the rapidly increasing size, complexity, and intractability of web-service networks gives rise to needs for effective method, to address the web-service composition problem. In this thesis, we have studied several issues centered around the web-service composition problem. First, we formulated the web-service composition problem in terms of AI planning and the network optimization problem. Second, we investigated publicly available web service sets using complex network analysis techniques. Third, we developed a novel web-service benchmark tool, WSBen, and a novel AI planning-based heuristic web service composition algorithm, WSPR. We verified the effectiveness and computational efficiency of WSPR with respect to the quality of solutions and solution time. In addition, we experimentally proved the scalability and robustness of WSPR in large and diverse service networks. Finally, we proposed an ontology-based framework using semantic web-service technologies to secure the reliable and large scale interoperability among the distributed entities of a manufacturing systems.

9.1 Contributions

In detail, some of the major contributions of this research are as follows:

1. **WSPR development**: As the emergence of service-oriented architecture provides a major boost for e-commerce agility, the number of available web services is rapidly increasing. However, when there are a large number of web services available and no single web service satisfies the given request, one has to "compose" multiple web services to fulfill the goal. Toward this problem, we have developed and implemented an AI planning-based web-service composition algorithm titled WSPR. The WSPR outperforms state-of-the-art AI planners, such as GraphPlan and Blackbox, in composing large-scale web services (in the range of 1,000 to 50,000) with respect to effectiveness, computational efficiency, scalability, and robustness. WSPR won the first runner-up award in the ICEBE05 web-service contest. The new release of WSPR will be featured with semantic matching support and will be submitted to the third 2006 web-service challenge[1].

2. **WSBen development**: As a growing number of web services are available on the web and in organizations, finding and composing the right set of web services has become even more important. As a result, in recent years, a plethora of research work and products on web-service discovery and composition algorithms have appeared. Despite all these efforts, there have been very few test environments available for evaluating such algorithms and software. To address these needs, we have designed and built WSBen, a web-service discovery and composition benchmark tool. The WSBen has two main characteristics: (1) Flexible web services matching framework, and (2) Diverse web-service network models, including the Erdos-Renyi random graph model, the Newman-Watts-Strogatz small-world network, and the Barabasi-Albert scale-free network. Besides the test set generation, WSBen provides a set of functions to simplify the benchmark process, such as exporting test sets into AI planner files including PDDL and STRIPS and generating test requests automatically.

3. **Analysis of existing web services**: We observed the network properties of

[1]The third 2006 web-service challenge is co-located with the Conference on Electronic Commerce (CEC) and the Conference on Enterprise Computing, E-Commerce and E-Services (EEE). For detail, enter the following site. http://insel.flp.cs.tu-berlin.de/wsc06/. WSPR was placed third in this challenge

public web services and ICEBE05 test sets. From the observation we discovered that public web services show similar features as the world wide web in terms of the small-world network and scale-free network properties. This can be inferred because the public web-service networks have grown in a similar manner as the Internet. On the contrary, the ICEBE05 web service networks are highly artificial so that their underlying topologies form tree structures but all arcs are directed toward the same direction without cycles. This is like business workflows, and the isolated sub-networks are uniformly partitioned so that the size of sub-networks are similar and the number of edges per node is uniformly distributed. This observation results in three rewarding lessons: (1) WSC problems can arise in diverse scenarios. Nevertheless, they can be captured by investigating which network topology would fit into their web-service networks; in particular, either small-world network or scale-free network; (2) The WSC algorithms must run on diverse application areas characterized by the underlying network properties; (3) Understanding the structural properties of networks often help gain better insights and develop better algorithms. As a potential future work, we can make another search heuristic algorithm to address WSC problems by exploiting the underlying network structure. For example, hub parameters (or hub web services) can serve important roles on the inter-connections between web services.

4. **Application of Semantic Web-service Composition in Manufacturing**: We proposed an ontology-based framework using semantic web-service technologies to secure the reliable and large scale interoperability among the distributed entities of a manufacturing systems, namely: a design firm, manufactures, and third party logistics providers. We reviewed existing semantic web service technologies and proposed an approximate scenario that forms the motivating base for our proposed framework. Currently, this work is only at the conceptual stage, with a focus on proposing a software design rather than implementing the proposed framework. However, we are considering the extension of WSPR by adding the OWL-DL based reasoning service in order to implement the suggested ontology-based framework.

9.2 Limitations and Assumptions

A more critical analysis of this work can be carried out by discussing limitations and assumptions of this work.

1. $c(w) = 1$: We assumed that for all web services w, the invoking cost $c(w)$ is 1 identical. In a real system, however, each web service may have distinct $c(w)$. $c(w)$ could be determined based on real market surveys, or using a pricing model specific to web services, suggesting that $c(w) = 1$ can be relaxed such that $c(w) > 0$. This relaxation has an impact on WSPR, because current WSPR's heuristic measures the contribution of a web-service based on how many of its output parameters overlap with a sub-goal. The modification of WSPR required to address this relaxation can be carried out by doing the following:

   - In the forward search step of WSPR, Line 7 in Algorithm 1 must be modified to follow the recursive Equation 6.1 to obtain $g_{r^i}(p)$.
   - In the backward search step of WSPR, web services in *wSpace* (Line 3 in Algorithm 2) must be sorted according to the cost for achieving them, and then measuring $h_{sg}(w)$ is applied to the web services with minimum cost.

2. Composition based on open-loop control: WSPR carries out the sequence of web-service compositions irrespective of the consequences, meaning that there is no feedback to constantly collect data from the previous consequences and pass the data to WSPR for better decision making afterward. We can extend current WSPR to combine a unsupervised learning mechanism (e.g., BootStrip [87]), through which WSPR can learn knowledge about problem domains, such as space, time, action, objects, causality.

3. Single criterion for determining effectiveness: $\#W$, the total number of web services in a solution is used as single criteria for measuring effectiveness of WSC algorithms. However, we can use a combination of multiple criteria for evaluating WSC algorithms. For example, we can formulate the objective of a WSC problem to maximize the reliability of a composite service as well as to minimize the total cost of invoking the composite service.

4. Exact parameter matching approach: When WSPR determines whether two web services are matched or not, it investigates whether their parameter names are the same or not, meaning that WSPR just relies on the exact parameter matching scheme. WSPR can be extended to use other parameter matching schemes such as the approximate and semantic matching schemes as described in Chapter 3.2.

9.3 Future Research

First, we can extend current work to address issues discussed in Chapter 9.2. Furthermore, some of the possible future research extensions are as follows:

1. **Web Services Monitoring and Diagnosis**: Self-healing software is one of the challenges for information science [102]. We can apply the self-healing concept to composing web services. In this research, we can expect following contributions:

   - Monitoring, detection and diagnosis of anomalous situations due to functional or non-functional errors (e.g., Quality of Service).
   - Repair/reconfiguration, thus guaranteeing reliability and availability of web services.
   - Methodologies and tools for helping design services such that effective and efficient diagnosability and repairability are guaranteed during execution.
   - Demonstration of these results on real applications.

   In fact, the work proposed by the WS-Diamond project [124] aims to add diagnostic capabilities to web services, in order to build self-healing workflows. This research issue is timely because the progress in model-based diagnosis and the emergence of web service standards that occurred in the last decade have reached sufficient foundational advances to make the proposed research feasible. In one of our previous research projects, we proposed a methodology titled MISQ[2] to analyze initial WSC problems and obtain optimized parame-

[2]For details, see Appendix C.

ters for the composition design. In MISQ, we use UML to design agent-based business processes and two formal modeling schemes, such as Stochastic Process Algebra (SPA) and Generalized Stochastic Petri Nets (GSPN), for the design analysis purpose. We believe that MISQ can be extended by adding the soundness for automatic execution, while contributing to this web service monitoring and diagnosis research field.

2. **Semantic web-service composition**: The web was originally created to enable the sharing of information among scientists. It has since evolved to extend its applications, including governments, businesses, and individuals, to make their data web-accessible. However, the majority of today's data on the web are "understandable" only to humans or custom-developed applications. The semantic web is introduced to address this issue [14, 15]. It is defined as an extension of the "existing" web in which information is given a well-defined meaning. The ultimate goal of the semantic web is to transform the web into a medium through which data can be shared, understood, and processed by automated tools [111]. The semantics of web services are crucial to enabling the automatic service composition. It is important to ensure that selected services for composition offer the "right" features. Such features may be syntactic (e.g., parameters included in a message sent or received by a service). They may also be semantic (e.g., the business functionality offered by a service operation or the domain of interest of the service) [72]. To help capture web services' semantic features, we can use the concept of ontology. In Chapter 8, we proposed an ontology-based framework using semantic web-service technologies to secure the reliable and large scale interoperability for the supply chain domain. However, this work is only at the conceptual stage, with a focus on proposing a software design rather than implementing the proposed framework.

   An ontology is a shared conceptualization based on the semantic proximity of terms in a specific domain of interest. Ontologies are increasingly seen as a key to enabling semantic-driven data access and processing. Because we expect that ontologies play a central role in the semantic web, we can take into account a research focused on extending syntactic service inter-

operability to semantic inter-operability.

3. **Template-based composition of semantic web services**: Our research on the web-service composition has been focused on the generic case where no domain knowledge exists, such as template structures to guide the planner to find a composition. In practice, however, workflow templates are used to design executable workflows in many applications, such as B2B applications, scientific grid applications, the intelligent distributed manufacturing [61], and dynamic supply chains. These applications can provide business rules that can be encoded using templates [103]. In particular, we can explore the following tasks in this research field:

    - Highlighting the most important features related to describing workflow templates, with specific focus on particular application (e.g., the intelligent distributed manufacturing).
    - Presenting algorithms for matching and ranking potential services that can be bound to abstract activities in a workflow as concrete services.
    - Suggesting AI planning formalism to combine the template description and the matching/ranking algorithm together.

4. **Multi-agent based web-service composition**: It is believed that the workflow model is not good enough to represent the composition of web service because the components of web services can be executed concurrently [10]. In other words, describing compositions of web services as a workflow seems harder than describing them as a multi-agent system, where concurrent execution of services can be easily represented. Thus, research is necessary to describe the composition of web service into a multi-agent framework, and monitor and diagnose the behaviors of web-service agents before launching the composition in a real case scenario.

Appendix A

WSBen with yTS framework

In Chapter 5, we used WSBen to generate test sets that were later used for extensive testing to validate the effectiveness and efficiency of WSPR. In those cases, WSBen uses a test generation scheme, $xTS = \langle |J|, Gr, \eta, M_p, |W| \rangle$, where Gr is bounded by one network model from three alternatives; (1) Barabasi-Albert model; (2) Newman-Watts-Strogatz model; and (3) Erdos-Renyi model. However, we are not assured that those complex and random network models can fit all possible cases that may arise when trying to design the parameter cluster network, G_{cl}. To fill the possible gap if we use only xTS, WSBen provides a more flexible test set generation framework named yTS. In this Appendix, we introduce yTS framework in detail, and use it to generate test sets that will be applied to extensive experiments for testing the performance of WSPR and other AI planners.

A.1 Flexible benchmark generation scheme

yTS framework is defined as a five tuple: $yTS = \langle J, Co, \eta, Rl, |W| \rangle$. Provided that the first four tuple are grounded, one can build a parameter cluster network, $G_{cl}(V_{cl}, E_{cl})$. If $i \in V_{cl}$ contains a small number of parameters but has a large incoming or outgoing edge degree, we refer to i as a hub cluster. In detail, each tuple of yTS is explained as follows:

(1) J is the set of clusters; $|J|$ denotes the total number of clusters.

(2) $Co(j)$ is the co-occurrence probability of parameters in a cluster $j \in J$. $Co(j)$

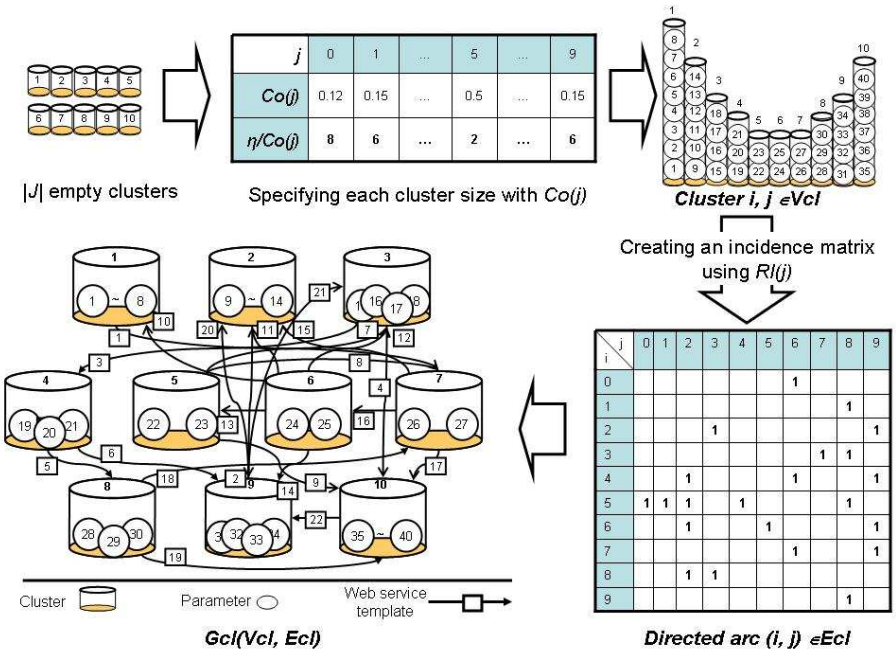


Figure A.1: G_{cl} generating process from $\langle 10, bell(0.3, 0.5), 1, bell(0.3, 0.5), |W| \rangle$

is bound by one distribution selected from following alternatives:

- *skewed* : $skew(j; \alpha, \beta) = \beta \exp^{(-\alpha \times j)}$
- *bell* : $bell(j; \alpha, \beta) = \beta \exp^{(-\alpha \times ABS(j - |J|/2))}$
- *uniform* : $uni(j; \alpha) = \alpha$

Where, α and β are constants given by users, such that any values of the selected alternative distribution must reside between 0 and 1.

(3) η is the parameter condense rate. This value is used to specify the size of a cluster. That is, the total number of parameters in Cluster j is $\eta / Co(j)$.

(4) $Rl(j)$ is the association distribution of a cluster $j \in J$, and represents the rate of outgoing edge degree of Cluster j. $Rl(j)$ shares the alternative distributions of $Co(j)$. For example, $Rl(j) = uni(j; 0.2)$ means that each cluster in G_{cl} has outgoing edges incident to 20% of other clusters in V_{cl}.

(5) $|W|$ denotes the total number of web services in a test set.

We can illustrate how a G_{cl} can be derived from a yTS specified simply by $\langle 10, bell(0.3, 0.5), 1, bell(0.3, 0.5), |W| \rangle$. The process is shown in Figure A.1 and explained in detail as follows:

1. $|J|$ size of empty clusters are generated.

2. The total parameter number of each cluster is specified by using Co and η. For example, Cluster 5 has two parameters because $1/[0.5 \times \exp^{(-0.5 \times ABS(5-|10|/2))}]$ ≈ 2, where η is 1 and Co is *bell* with $\alpha = 0.3$ and $\beta = 0.5$. It must be noted that the parameter names are automatically generated (e.g., "1" or "par1") and thus do not contain any semantics.

3. An arc incidence matrix to define the connectivity between clusters are created using Rl. For example, Cluster 5 has five outgoing edge degree because $Rl(5) = bell(5; 0.3, 0.5) = 0.5$ so that Cluster 5 is connected to 50% of the other clusters in V_{cl}.

4. Once G_{cl} is built, we can generate $|W|$ size of web services by taking the following two steps:

   (a) Randomly choosing $\langle i, j \rangle \in E_{cl}$ that is a web-service template.

   (b) Generating input parameters from cluster i, according to $Co(i)$, and output parameters from cluster j, according to $Co(j)$.

   Note that each parameter in one cluster can map into either an input parameter or an output parameter. In the case that no parameter is generated, and instead a dummy parameter 'S' and 'T' is filled in the position of input and output parameters, respectively.

For the experimental validation, we build three test set frameworks by materializing yTS as follows:

(1) uTS: $\langle 100, uni(0.2), 1, uni(0.2), |W| \rangle$

(2) bTS: $\langle 100, bell(0.3, 0.5), 1, bell(0.3, 0.5), |W| \rangle$

(3) sTS: $\langle 100, skew(0.5, 0.5), 1, skew(0.5, 0.5), |W| \rangle$

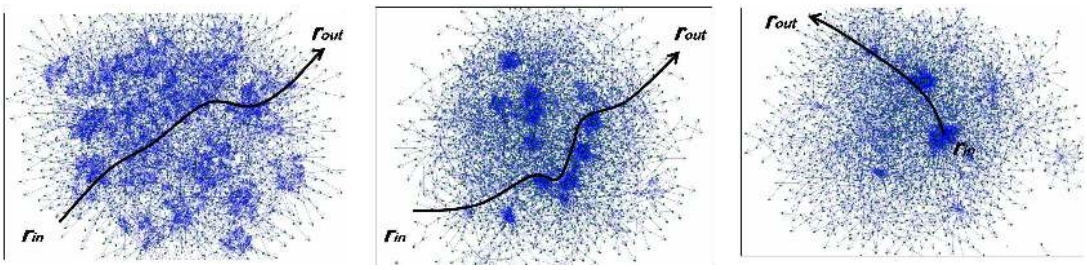


Figure A.2: G^f_{op} at $|W| = 1,000$. (left) uTS. (center) bTS. (right) sTS

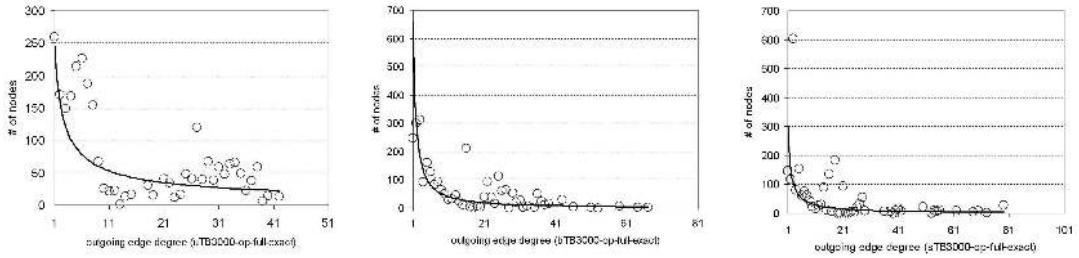


Figure A.3: Outgoing edge degree distribution of G^f_{op} at $|W| = 3,000$. (left) uTS. (center) bTS. (right) sTS

For each framework of uTS, bTS, and sTS, its unique G_{cl} is generated, and based on it, 9 different size of test sets are instanced by varying $|W|$ as 1,000, 3,000, 5,000, 7,000, 9,000, 11,000, 15,000, 20,000, and 30,000 respectively. Thus, 27 test sets are generated. Each test set has one test request. In addition, in order to achieve necessary results for the statistical analyses, we repeat this generation process twice. Consequently, 81 test sets are prepared. A test request is constructed, such that r^i is all atomic parameters contained in the cluster 0, and r^o consists of the first five largest parameters in terms of $g_{r^i}(p)$[1], which corresponds to the cost taken to reach p in the forward search over the parameter space. Thus, r^o are farthest away from r^i in a parameter space.

Figure A.2 shows G^f_{op} when $|W| = 1,000$. Note that uTS and sTS have hub nodes (i.e., hub web services) which have a large outgoing edge degree. In fact, at the small size, hub nodes are only glimpses, but as the test set size increases, they become more distinguishable and identifying. A directed solid line in each graph represents an artificial solution trajectory from r^i to r^o, assuming that there would

[1] $g_{r^i}(p)$ is the cost of achieving $p \in P$ from r^i by the forward search over the parameter space. A more detailed explanation will be given in Chapters 6 and 7

be a solution consisting of full-matching operations.

Three graphs in Figure A.3 show the outgoing edge degree distributions of G^f_{op} for each of uTS, bTS, and sTS when $|W| = 3,000$. They are differentiated by each having a different range of x-axis and y-axis, such that uTS has 1-42(x) and 0-250(y); bTS has 1-67(x) and 0-300(y); and sTS has 1-79(x) and 0-600(y). Thus, sTS has its distribution more skewed than uTS and bTS. This is inferred because sTS is characterized by the exponential distribution, which is more skewed than uniform and bell distributions that characterize bTS and uTS, respectively.

Figure A.4 shows the outgoing edge degree distribution of G_p for each of uTS, bTS, and sTS. uTS has a bell shape distribution. The shape is expected, since both Co and Rl have the same uniform distribution, and so theoretically all parameters have the same chance of occurring. Furthermore, since $|W| = 3,000$, it obeys the Law of Large Numbers statistically so that the shape of distribution converges to a bell or normal curve. Regarding the distributions of bTS and sTS, they follow a Zipf-like distribution because both Co and Rl have the same skewed distribution: Most parameters have very low chance of occurring but only a small number of parameters contained in hub clusters have a high possibility of occurring. These observed differences between test sets become more significant as their test-set size increase.

The complexity of test sets is varied in different dimensions which are partitioned as follows:

- Test set and parameter size: we varied $|W|$ of uTS, bTS and sTS in the same scale from 1,000 to 30,000. However, the total parameter number $|P|$ of uTS, bTS, and sTS is determined differently. As mentioned before, $|P|$ of each framework is set after binding the first four tuple of yTS. Since sTS and bTS use the exponential function in Co and Rl, their $|P|$ grows exponentially as $|J|$ increases. On the contrary, $|P|$ of uTS does not change even though $|J|$ increases, because uTS uses the uniform function in Co and Rl. Note that the search space (i.e., state space) is exponential to the parameter space.

- The size of alternatives of a web service w: suppose that there is a set of web service W_A, such that for $w_a \in W_A$, $w^i_a \subseteq w^i$ and $w^o_a \supseteq w^o$. It is clear that

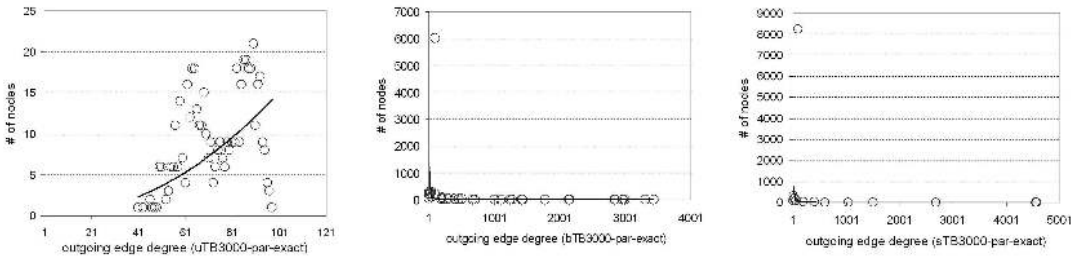


Figure A.4: Outgoing edge degree distribution of G_p at $|W| = 3,000$. (left) *uTS*. (center) *bTS*. (right) *sTS*

$w_a \in W_A$ can replace w_i. *uTS* has a greater chance to find larger alternatives for a given web service than other frameworks, because *uTS* connects clusters in such a way that the connectivity between clusters is uniformly distributed. Thus, the computation cost to search the right web services in *uTS* can be very low.

- The trajectory between r^i to r^o: Figure A.2 suggests artificial trajectories for each of *uTS*, *bTS*, and *sTS*. A trajectory can be a solution path, or simply, a sequence of web services starting from r^i to r^o. For example, in *sTS* case, the solution trajectory may start in the darkest dot in the center where many of web services are concentrated with their input parameters selected from Cluster 0. Note that we set r^i with the parameters in Cluster 0. Since the trajectory starts from the center of G^f_{op}, the length of the trajectory becomes short compared to those of *uTS* and *bTS*, where the trajectories are created between two parameter clusters that are the farthest away from each other.

In the experiments described below, we evaluated the performance of three prominent AI planners and our proposals: Graphplan [19], Blackbox [55], IPP [51], WSPR, and WSPR w/o heuristic. The experimental environment is the same as the experiments conducted in Chapter 7.

A.1.1 Result

The following set of experiments deals with *uTS*, *bTS*, and *sTS*. The results involving *uTS* are shown in Figure A.5. In some tests, Graphplan and IPP fail to solve problems when the test set size increases. Regarding *Time*, all competitors

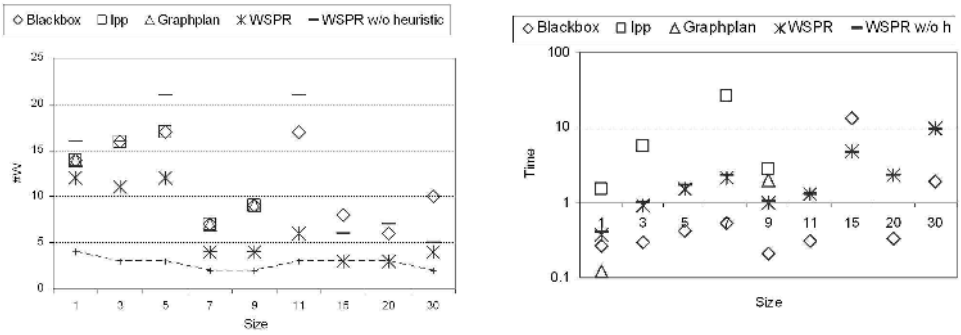


Figure A.5: Results of uTS in first replication. (left) $\#W$. (right) *Time*.

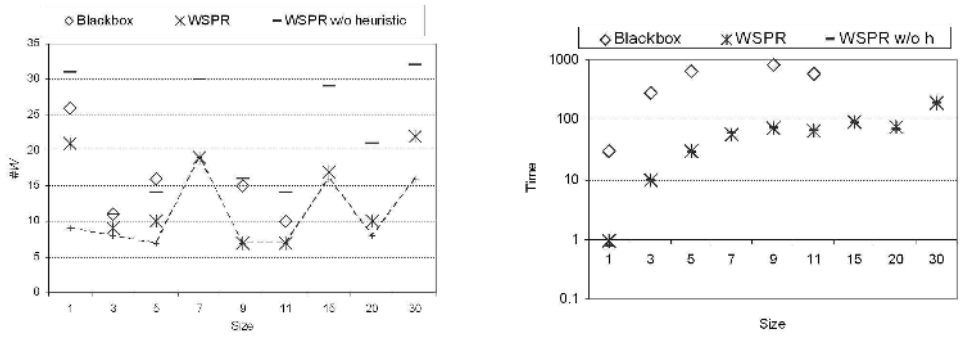


Figure A.6: Results of bTS in first replication. (left) $\#W$. (right) *Time*.

solve requests within a reasonable time frame. The results concerning bTS and uTS are shown Figures A.6 and A.7, respectively. Graphplan and IPP fail for all cases. Regarding *Time*, both WSPR and WSPR w/o heuristic scale up smoothly and clearly overrun Blackbox.

For a better understanding of the difference between competitors in terms of $\#W$, we can conduct a statistical analysis on $\#W$ results data. First, we ranked competitors with respect to $\#W$, such that the competitor with the less # is ranked higher. Note that some competitors, in particular, Graphplan and IPP did not generate solutions in some cases, making simple average or sum over three replications inappropriate. Remember that we generated two additional replications on each of the 27 test sets. The statistical analysis reveals that significant differences exist between the mean rank for many of the algorithms. An ANOVA analysis showed significant differences across algorithms for the pooled data ($p < 0.001$). The low

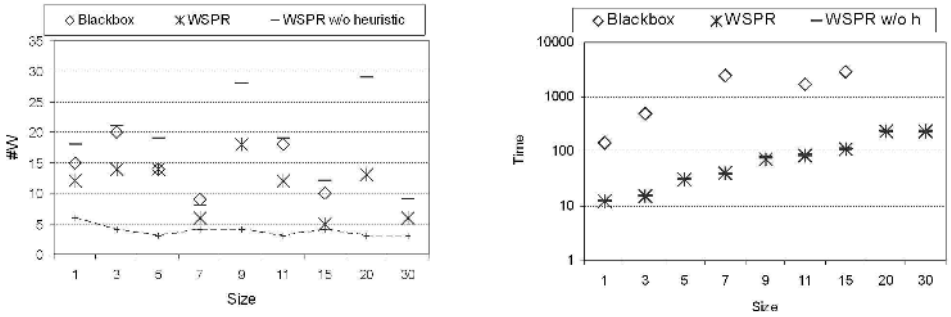


Figure A.7: Results of sTS in first replication. (left) $\#W$. (right) *Time*.

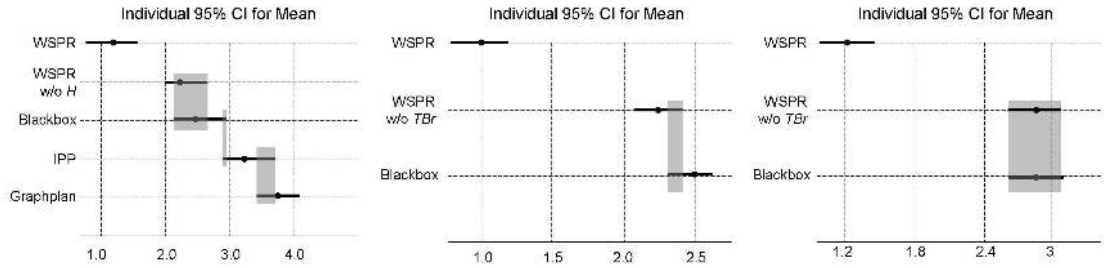


Figure A.8: 95% confidence intervals (CI) of the mean rank for $\#W$ between algorithms. (left) uTS . (center) bTS. (right) sTS

p-value indicates a strong probability that at least one algorithm's mean rank is significantly different from the others. Therefore, in order to specify which algorithms differ from the others and which do not, a multiple comparisons procedure with the Tukey's method[2] [48] was performed.

Figure A.8 shows the 95% confidence intervals of the mean rank for each algorithm. In the figure, an overlap between any two or more intervals indicates that there is not a statistically significant difference between the corresponding means. The center dots in the intervals represent the mean values. From this analysis, WSPR has a considerable performance difference from other AI algorithms. WSPR also shows a significant difference statistically with WSPR w/o heuristic in all cases.

Figure A.9 supports these statistical results by illustrating solutions generated by WSPR and Blackbox, where the test case is bTS with $|W| = 9,000$ in the first

[2]Given multiple data sets, the Tukey's method is applied to consider all possible pairwise difference of means between the data sets simultaneously

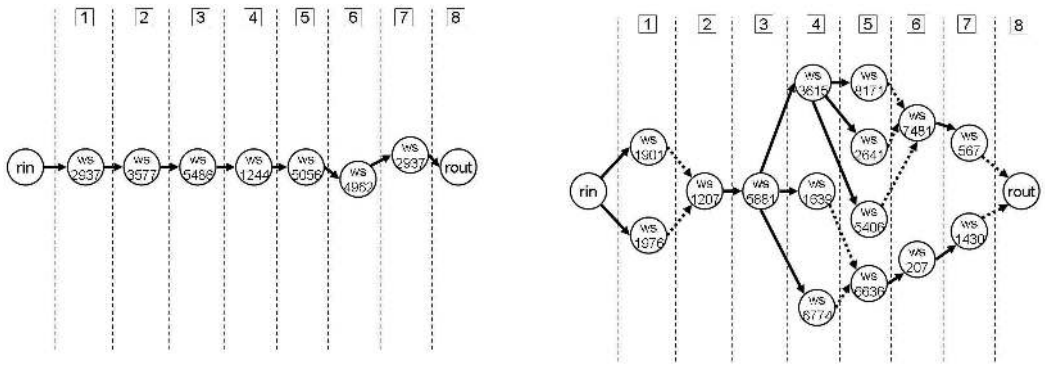


Figure A.9: Solutions to bTS at $|W| = 9,000$ in first replication. (left) WSPR. (right) Blackbox

replication. In the figure, a solid line represents the full-matching operation while a dotted line represents the partial-matching operation. This comparison shows that WSPR is more capable of finding a solution consisting of a sequence of fully matching web services than Blackbox.

Summary: The above experiments coincide with the results obtained in Chapter 7, where WSPR tends to perform better than the AI planners in all cases, especially, when a web service network topology is skewed and the number of parameters is very large. We also prove that WSPR heuristic, based on the strategy in support of locating a fully-matching web service first in the regression search, is in effect. Furthermore, the experiments show that the two stage approach of WSPR results in the scalability of WSPR, because *Time* of both WSPR and WSPR w/o heuristic do not blow up exponentially in the increasing test set size. Lastly, the experiments assure the usefulness of the suggested test set generation framework, because by using the frameworks, we can generate such difficult test sets through which we can distinguish the algorithm performances in terms of $\#W$ and $Time$. Additionally, we can investigate in detail how they operate in different environments (e.g., small versus large sizes; skewed versus uniform distributions; short solution length versus long solution length).

Appendix B

Additional Discussion on ICEBE05

We discovered that the ICEBE05 [54] test sets have interesting characteristics with respect to their problem complexities. This discovery is presented in this Appendix.

B.1 Discussion on Test Sets

According to the description from the ICEBE05, the construction of test data varies in different dimensions as follows:

- Minimal $\#W$[1]: they provide test sets such that some test sets have their minimal $\#W$ between 2 and 4 (test set names starting with Composition-1), and other test sets have their minimal $\#W$ between 6 to 8 (test set names staring with Composition-2).
- The number of web services in a test set (i.e., $|W|$): $|W|$ of test sets is one among 3,356 and 5,356 and 8,356. $|W|$ of test sets can be identified from their test set name because 20, 50, and 100 figures included in the test set name indicate 3,356, 5,356, and 8,356, respectively.
- Number of input and output parameters: the size of input and output parameters of web services is differentiated in the range of 4-8, 16-20, and 32-36.

[1] $\#W$ is the number of web services in a solution. It is counted to measure the quality of the found solution, or the effectiveness. See Chapter 7 for details.

The test set names in ICEBE05 indicate the combination of the parameters by labeled Composition-1 and Composition-2, followed by the test set size and the number of parameters. For example, "Composition1-50-32" represents that: (1) it is the service Composition-1 with a minimal $\#W$ between 2 to 4; (2) $|W|$ is 5,356; and (3) the number of input and output parameters is in the range of 32 to 36.

Both Composition-1 and Composition-2 have nine test sets that each has 11 queries. As our experiments using WSPR show, solving requests in Composition-2 is harder than Composition-1 in terms of $\#W$ and $Time$. For example, $\#W$ in Composition-1 is in the range of 2 and 4 as shown in Table B.1, while $\#W$ in Composition-2 is in the range of 6 and 8 as shown in Table B.3. In terms of Time[2], Composition-1 is in the range of 49∼703, while $Time$ in Composition-2 is in the range of 96∼2281. This implies that we need more resources (i.e., longer Time and larger $\#W$) in Composition-2 than Composition-1. We also checked how resource consumption is divided into two phases of WSPR. For this purpose, we denoted $fTime$ to represent the time consumed in forward search, and $bTime$ to represent the time consumed in backward search; $Time = fTime + bTime$. Then, we recorded $fTime$ and $bTime$ in each search.

B.1.1 Composition-1

From the perspective of $\#W$ as shown in Table B.1, we discovered that the 11th request was the simplest while the 7th through 10th requests were the hardest; the 11th request was solved with just one web service, while the 7th through 10th requests required as many as four web services. From the perspective of test set's difficulty level, as shown in Table B.2 and (a)-(c) of Figure B.1, we found that 20-4 test set is the simplest, while 100-32 test set is the most difficult. These results are closely related to the size of the test sets. In other words, since 20-4 and 100-32 have 3,356 and 8,356 web services, respectively, the search space of 100-32 becomes much bigger than that of 20-4.

In terms of Time, the 9th request in 100-32 test set takes the longest (i.e., 703 ms). When it comes to resource consumption in two different phases, most resource consumption was assigned to the forward reasoning stage. No $bTime$ exceeded 3

[2] $Time$ measures how long an algorithm takes to find a solution. It is counted to measure the computational efficiency.

Table B.1: Comparison of $\#W$ over requests (Composition-1)

Request No.	1	2	3	4	5	6	7	8	9	10	11
$\#W$	2	2	2	3	3	3	**4**	**4**	**4**	**4**	**1**

Table B.2: Comparison of 11 requests over 9 test sets in terms of $fTime$ and $bTime$ (Composition-1)

Request No.	*Time*	20-4	20-16	20-32	50-4	50-16	50-32	100-4	100-16	10-32
1	$fTime$	46.9	109.9	187	46.9	125	202.9	94	219	344
	$bTime$	16	2	2	15.1	2	31	14.9	2	14.9
2	$fTime$	**47**	110	172	62	171.9	217.9	109.9	203	343.9
	$bTime$	**2**	2	2	2	2	2	2	2	15
3	$fTime$	62	109.9	171.9	62	141	217.9	110	219	358.9
	$bTime$	16	2	2	2	2	16	2	2	2
4	$fTime$	62.9	140.9	250	108.9	202.9	328.9	139.9	312	500
	$bTime$	2	2	2	2	2	2	2	2	2
5	$fTime$	62.9	140	234.9	78	187.9	328.9	139.9	296.9	671.9
	$bTime$	2	2	15.1	16	15.1	2	2	16	2
6	$fTime$	62.9	141	265.9	109.9	250	258.9	157	297	515
	$bTime$	15.1	2	15	2	2	116.1	2	2	2
7	$fTime$	78	187	328	109.9	250	406	187.9	391	639.9
	$bTime$	16	2	2	15.1	2	2	2	2	16
8	$fTime$	92.9	203	328	125	328	421	171.9	389.9	671.9
	$bTime$	16	2	2	2	16	2	2	2	2
9	$fTime$	92.9	218.9	328	109.9	250	406	203	375	**671**
	$bTime$	16	2	2	2	16	2	2	2	**32**
10	$fTime$	92.9	187	327.9	108.9	233.9	453.9	187	407	657
	$bTime$	2	2	2	2	16.1	2	2	14.9	2
11	$fTime$	77.9	187	359	78.9	156	344	92.9	187.9	312
	$bTime$	2	2	2	2	2	2	2	2	2

ms as shown in Table B.2. This is explained with two reasons:

1. All requests are solved by linearly sequential full-matching composition. Consequently, in the test sets in the ICEBE05, WSPR can save $O(|W|log|W|)$ computations because WSPR does not need to measure $h_{sg}(w)$ at all. Instead, WSPR can just select one web service listed in $wSpace$. In other words, while the first forward reasoning step keeps the computational cost as high as $O(|W|^2|P|)$, the second regression planning step brings the cost down to $O(|W|)$.

2. The solution length obtained is very small. For example, in 100-32 test set, the 10th request can be solved with four web services. That is, regression

Table B.3: Comparison of $\#W$ over requests (Composition-2)

Request No.	1	2	3	4	5	6	7	8	9	10	11
$\#W$	6	6	6	7	7	7	**8**	**8**	**8**	**8**	**1**

planning reaches r^i starting from r^o in four iterative steps ($\ll |W|$).

B.1.2 Composition-2

Like Composition-1, from the perspective of $\#W$, we discovered that the 11th request was the simplest, but the 7th through 10th requests were the hardest, as illustrated in Table 5. While the 11th request requires just one web service, the 7th through 10th requests require eight web services.

Similar to Composition-1, from the perspective of the test set's difficulty level, we found that 20-4 is the simplest, but 100-32 is the most difficult as shown in Table B.4 and (d)-(f) of Figure B.1. In terms of $Time$, the 10th request in 100-32 takes the longest (i.e., 2280.9 ms = 2265.6 ms + 15.3 ms). When it comes to resource consumption in two different phases, most resource consumption was assigned to the forward reasoning stage. No $bTime$ exceeded 30 ms, as shown in Table B.4.

Summary: Although WSPR spends extra resources to maintain auxiliary structures such as $g_{r^i}(p)$ and $PD_{ws}(p)$ in the parameter space, overall we concluded that it is beneficial because WSPR can be more informed in the regression searching step due to the guidance of $g_{r^i}(p)$ and $PD_{ws}(p)$. However, most of the execution time of WSPR is spent in the forward reasoning stage due to the sheer number of web services to visit. Thus, in order to improve the overall speed of WSPR, better way is to be more informed about the parameter space need to be explored - which is a future research question to be addressed.

B.2 Discussion on the Request Complexity

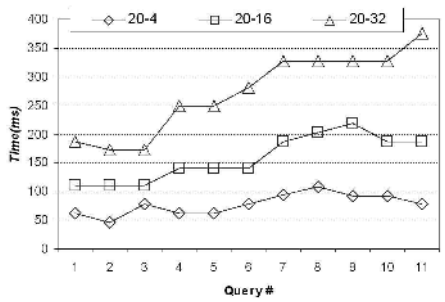
The WSC problem can be addressed by the linear planning approach in the AI community. The linear planning approach requires only AND operator. In the WSC context, the AND operator can be partitioned into "full-matching" oper-

Table B.4: Comparison of 11 requests over 9 test sets in terms of $fTime$ and $bTime$ (Composition-2)

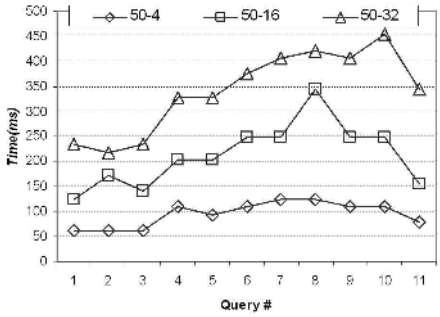
Request No.	*Time*	20-4	20-16	20-32	50-4	50-16	50-32	100-4	100-16	10-32
1	*fTime*	187.9	828.9	672	312	641	1093	500	1015.9	1734.9
	bTime	2	2	2	2	2	15.9	16	2	2
2	*fTime*	187	780.9	671.9	328.9	655.9	1141	516	1016	1733.9
	bTime	16	16	2	2	2	14.9	2	15	2
3	*fTime*	203	769.9	671.9	312.9	641	1094	500	1015.9	1717.9
	bTime	2	2	16	16	2	2	15	2	16
4	*fTime*	234.9	906	780.9	359.9	750	1312.9	594	1171.9	2000
	bTime	2	16	2	15.1	2	15	15.9	2	16
5	*fTime*	233.9	922	782	390.9	750	1282	594	1187	2000
	bTime	2	15	15	2	15.9	15	15	16	15.9
6	*fTime*	218.9	905.9	781	359.9	750	1250	592.9	1171.9	2000
	bTime	15	16	2	15.1	2	32	16	15.1	16
7	*fTime*	391	1030.9	890	421.9	842.9	1467.9	655.9	1328.9	2266
	bTime	2	16	15.9	16	16	15.8	32	2	16
8	*fTime*	264.9	1046.9	875	437	828	1421	657	1327.9	2266
	bTime	2	16	30.9	2	16	16	30	2	2
9	*fTime*	266	1046.9	875	421	844	1469	671	1327.9	2264.9
	bTime	2	2	16	2	16	2	16	16	2
10	*fTime*	250	1030.9	875	421.9	844	1546.9	671.9	1328.9	**2265.6**
	bTime	2	15.1	30.9	2	14.9	2	2	2	**15.3**
11	*fTime*	**94**	296.9	406	108.9	250	453	125	266	468.9
	bTime	**2**	15.1	2	2	16	2	2	2	2

ation and "partial-matching" operation. Interestingly, the EEE05 and ICEBE05 can be solved by using "full-matching" operation alone. That is, all requests can be satisfied by chaining web services in such a way that a predecessor web service can fully match the successor web service. Formally, suppose that Sol denotes the answer to a request r provided. Then, Sol is constructed such that $w_1^i \subseteq r^i$ and either $w_k^i \subseteq w_{k-1}^o$ or $(w_k^i \setminus r^i) \subseteq w_{k-1}^o$ for $k = 2, \ldots, |Sol|$. For example, the 2nd request of EEE05 can be solved by a chain of web services : `findMostRelevantStock` $\Rightarrow$ `getStockPriceMany` $\Rightarrow$ `performStockResearch` $\Rightarrow$ `purchaseOptimalStock`, such that

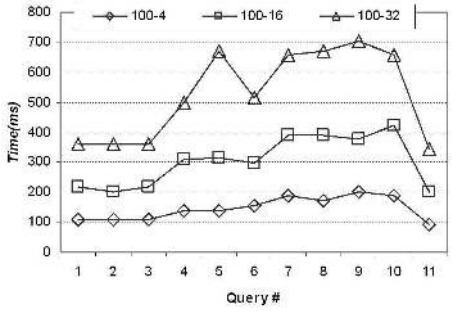
- `findMostRelevantStock`$^i \subseteq r^i$,
- `getStockPriceMany`$^i \subseteq$ `findMostRelevantStock`^{o},
- `performStockResearch`$^i \subseteq$ `getStockPriceMany`^{o},



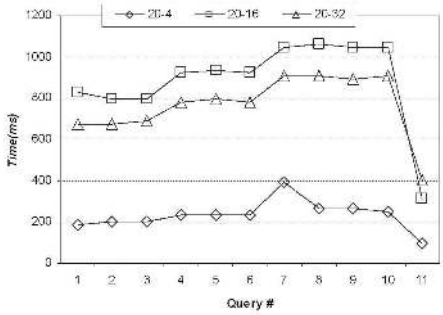
(a) Comparison of *Time* in Composition-1



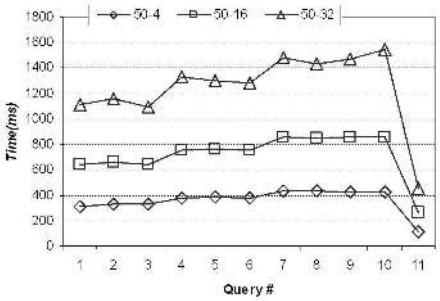
(b) Comparison of *Time* in Composition-1



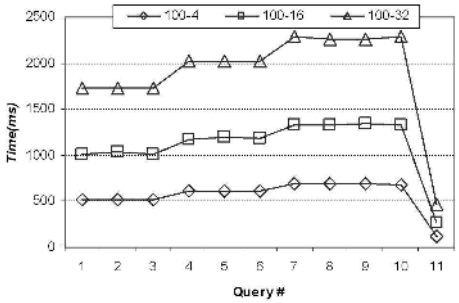
(c) Comparison of *Time* in Composition-1



(d) Comparison of *Time* in Composition-2



(e) Comparison of *Time* in Composition-2



(f) Comparison of *Time* in Composition-2

Figure B.1: Comparison of test sets in ICEBE in terms of *Time*

- `purchaseOptimalStock`$^i \subseteq$ `performStockResearch`^{o}, and
- $r^o \subseteq$ `purchaseOptimalStock`^{o}

Likewise, the 4th request in Composition2-100-32 of ICEBE05 can be solved by a chain of web services[3] : `s34a1827462` $\Rightarrow$ `s30a2162659` $\Rightarrow$ `s92a3046444` $\Rightarrow$

[3]In the original data set, `s34a1827462` is called `servicep34a1827462` (and others as well). We shortened them here for a simpler presentation.

s60a4328073 $\Rightarrow$ s19a0320284 $\Rightarrow$ s09a7063054 $\Rightarrow$ s58a4762157, such that

- $\texttt{s34a182746}^i \subseteq r^i$,
- $(\texttt{s30a2162659}^i \setminus r^i) = \{p76a0550626\} \subseteq \texttt{s34a1827462}^o$,
- $(\texttt{s92a3046444}^i \setminus r^i) = \{p15a6821354\} \subseteq \texttt{s30a2162659}^o$,
- $(\texttt{s60a4328073}^i \setminus r^i) = \{p54a2653485\} \subseteq \texttt{s92a3046444}^o$,
- $(\texttt{s19a0320284}^i \setminus r^i) = \{p30a6922384\} \subseteq \texttt{s60a4328073}^o$,
- $(\texttt{s09a7063054}^i \setminus r^i) = \{p53a0652249\} \subseteq \texttt{s19a0320284}^o$,
- $(\texttt{s58a4762157}^i \setminus r^i) = \{p67a9261643\} \subseteq \texttt{s09a7063054}^o$, and
- $r^o \subseteq \texttt{s58a4762157}^o$

Note that other queries in the EEE05 and ICEBE05 also take the same solution scheme (i.e., linear and sequential full-matching), as the examples described above. When an information gathering problem takes this special scheme, the problem can be solved quickly. In particular, if every step of a composition process requires only one parameter, it can be seen as a simple single source shortest-path problem. For example, in the 4th request in Composition2-100-32 of ICEBE05, the problem can be solved by finding a shortest-path conversely from the goal parameter, $r^o = p94a3566970$ to r^i by gathering $p67a9261643$, $p53a0652249$, $p30a6922384$, $p54a2653485$, $p15a6821354$ and $p76a0550626$, subsequently.

Appendix C

MISQ: A Framework for Automatic Implementation of Web-services Composition

In this Appendix, we present our previously proposed system, entitled MISQ [82], that aims at allowing users to analyze initial business processes and to obtain optimized parameters for implementing and monitoring their web-service composition.

C.1 Motivation

In Business Service Networks (BSN), by combining multiple, heterogeneous "services", one can establish new value-added business processes for further applications. In particular, web services have emerged as a popular means to describe the "services" that each vendor provides. Web services [114] are a piece of XML-based software interface that can be invoked over the Internet, and can be roughly viewed as a next-generation successor of CORBA or RPC technique. In such a setting, one of the key issues is how to generate, discover, compose, and optimize web services that are of interest. In this research, we especially focus on the problem of optimizing web-service composition and propose a novel methodology, MISQ, as a solution. Specially, we use UML to design agent based business processes, and two formal modeling schemes, Stochastic Process Algebra (SPA) and Generalized

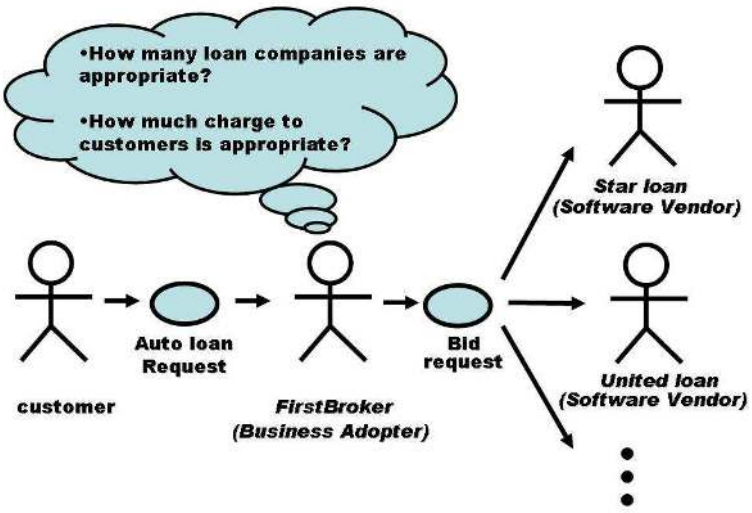


Figure C.1: Use case of FirstBroker example

Stochastic Petri Nets (GSPN) [93], to analyze initial business processes design and to obtain optimized parameters. Finally, we propose to use the Business Process Execution Language for Web Service (BPEL4WS) [7] as implementation artifacts for expressing the optimized business processes.

C.1.1 Motivating Example

Consider a scenario in a BSN's where the optimization of composed web services is a crucial issue. Suppose Bill opens an Internet-based auto loan brokerage company, $FirstBroker$, where he locates a loan with a low interest rate for customers who pay a nominal fee as a return. $FirstBroker$ uses web services from three loan companies, $StarLoan$, $UnitedLoan$, and $BestLoan$. Once $FirstBroker$ gets the customer's inquiry, it sends bid requests to the three loan companies using their web services, and forwards the lowest interest rate found to the customer. Whenever $FirstBroker$ sends loan rate requests to the loan companies, it has to pay a fee to each, as $FirstBroker$ is a business adapter and three loan web services are software vendors in the BSN. Furthermore, a customer pays a fee to $FirstBroker$ only if she is satisfied with the proposed rate and decides to make a contract with $FirstBroker$. In summary, Bill's profit model is the following:

- Profit model = (# of accepted proposals by customers × charge per customer) - (# of loan rate requests × # of loan companies × charge per loan rate request)

Suppose Bill agrees to pay $1 for each loan rate request to loan companies, while charging $10 to customers who eventually accept the proposed rate. The business is initially booming, attracting a large number of customers due to the fact that customers do not have to pay for initial inquiries, and pay $10 only after accepting. However, $FirstBroker$ eventually files a bankruptcy despite many customers submitting inquiries.

The scenario presented often occurs in combining and composing new services in BSN where a decision for parameters must be made to maximize profits. If Bill had chosen the correct number of web services (i.e., loan companies) and the proper service charge to customers, he would possibly remained in business. Like the case of $FirstBroker$, early identification of optimal values through formal analyses is particularly desirable, since the costs of changing the design at a later stage are much higher [68]. However, identifying optimal values when multiple web services are complicatedly inter-related is a challenging task, since in real applications, such parameters to consider can be many and non-trivial. Therefore, there is an imminent need for the methodology that systematically and mechanically helps to model, analyze, and optimize web-service compositions. For this solution, we propose MISQ in this research.

C.2 Overview of MISQ

As illustrated in Figure C.2, MISQ consists of analysis and implementation stages. Informally, the analysis stage runs as follows:

1. Design high-level UML diagrams such as state and sequence diagrams.
2. Transform high-level UML designs into a formal model in Stochastic Process Algebra (SPA) model.
3. Transform SPA into Generalized Stochastic Petri-Net (GSPN) model using steps suggested in the previous research [93].

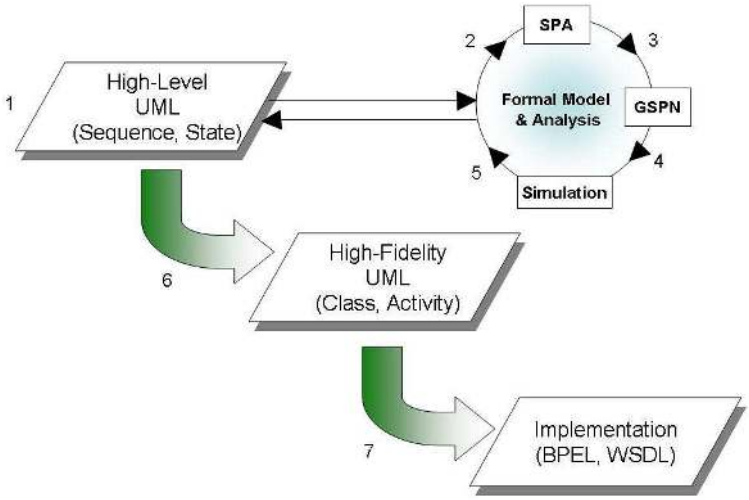


Figure C.2: Overview of MISQ

4. Perform analysis via simulation.

5. Based on simulation results, identify optimal parameters and design. If needed, steps 2-4 may be repeated.

The implementation stage is adopted from the waterfall model [6] of software development. It runs as follows:

6. Based on the optimized high-level design, produce a high-fidelity model consisting of class and activity diagrams.

7. From the high-fidelity model, generate implementation artifacts.

MISQ contributes to the following:

- A Petri-Net model for analyzing initial high level UML based designs, and the temporal and functional analysis for optimization can increase productivity and reliability of web service-based software systems in BSN's.

- A methodology for seamless integration of several languages or modeling tools (e.g., UML, SPA, GSPN, WSDL and BPEL), and a detailed example with a simulation result to illustrate the effectiveness of the proposed methodology.

C.3 Related Work

Our research integrates three different streams of work: (1) Deriving analysis model from UML; (2) Deriving implementation artifacts from UML; and (3) Transforming models from SPA to GSPN. To remedy the lack of verification and validation inherent in UML, some researchers have tried to translate UML into the process algebra [58]. The focus is on a sequence diagram where the objects are considered as π-calculus processes and messages are represented as actions among these processes. Despite the inherent semi-formality, UML has a strong descriptive power for high-level, as well as high-fidelity modeling [6]. Among UML diagrams, state diagrams and sequence diagrams are sufficient to represent the high-level model. On the other hand, a high-fidelity model is capable of representing the details of implementation artifacts. Usually, a high-fidelity model can be expressed with class and activity diagrams of UML. The mapping from high-fidelity model to corresponding implementation artifacts is provided using UML 1.4 profile and BPEL4WS [7] as implementation artifacts [6]. Comparisons between GSPN and SPA with different perspectives are given [34]. In our proposal, we use both GSPN and SPA as an analysis model to optimize web service composition. As dynamic discovery of web services and composition problems, run-time adaptability of a composed process is another research issue in this area. METEOR-S [27] project has addressed this issue for workflows. This allows the process designers to bind web services to an abstract process, based on business and process constraints, and generate an executable process. Proteus [45] was suggested as a framework that consumes a user request to compose a plan that incorporates available web services, and execute the plan seamlessly. Finally, in both METEOR-S and Proteus, the user defines a composition at design-time in the former, and in the latter service adopting is processed at execution time by assembling available web services. These compositions are pattern based and other researches have also addressed this approach [13]. Besides the dynamic composition approach, automatic composition of web services is a challenging research problem. This is due to difficulty of mapping user needs to a collection of correlated services, where their interim outputs can satisfy each other's input requirements and the final delivery meets the user demands. Finally, matching interfaces between each web service in the collection is a problem. If only

syntactic matching is permitted, the problem can be formulated into a special directed graph shortest path problem [84]. On the other hand, semantic interface matching is expected to be crucial to automatically compose new services due to the increasing number and heterogeneity of available web services. Interface-Matching Automatic Composition technique [131] incorporates the use of the web-service ontology to find matching web services. There is the emerging consensus that the ultimate challenge is to make web services automatically tradable and usable by artificial agents in their rational, pro-active interoperation on the next generation of the web [40]. It may be solved by creating effective frameworks, standards and software for automatic web-service discovery, execution, composition, interoperation and monitoring [71]. In the industries, only initial and partial solutions of the ultimate problem are provided. Existing de-facto standards for web service description (WSDL) [115], publication, registration and discovery (UDDI) [113], binding, invocation, and communication (SOAP) [112] provide merely syntactical capabilities and do not completely solve the ultimate challenge. More recent research and standardization activities of DARPA DAML community resulted in offering semantic service markup language DAML-S [8] based on RDF platform.

C.4 MISQ Methodology

MISQ is based on various models (UML, SPA, GSPN, BPEL, and WSDL) as well as the transformation procedures between models. In the interest of space, here we present a brief overview of SPA and GSPN.

Definition C.4.1 (SPA)**.** *Stochastic Process Algebra (SPA) is described by the following grammar, [34].*

$$P := Stop \quad | \quad (a, \lambda).P \quad | \quad a.P \quad | \quad P + P \quad | \quad P \|_s P \quad | \quad P \backslash S \quad | \quad Q \qquad \text{(C.1)}$$

where a variable P, Q,... denotes the process variables, while S is a set of synchronization actions. The intuitive meaning of these elements is:

- *Stop* denotes the halting process.
- The process $(a, \lambda).P$ models a delayed process that performs the action a

with delayed rate λ and then behaves as process P.

- The process $a.P$ models an immediate process that performs the action a without any delay and then behaves as process P.
- The choice operator '+' is used to model alternative behavior.
- The parallel operator '$\|_s$' models the parallel execution of two processes which have to synchronize in actions within the set of synchronizing actions S.
- The hiding operator '\' is used for declaring actions as internal, and is often used to abstract away from internal events.

Definition C.4.2 (GSPN)**.** *Generalized Stochastic Petri-Net (GSPN) [34] is defined as a 5-tuple* (PL, T, W, M_0, L)*, where:*

- PL *is a finite set of places.*
- T *is a finite set of transitions partitioned into two sub-sets* T_I *(immediate) and* T_D *(delayed) transitions, where, transitions* $t \in T_D$ *are associated with delayed rate* λ*.*
- $W \in (PL \times T) \cup (T \times PL)$ *is a set of directed arcs (i.e., flow relation).*
- $M_0 : PL \to \{0, 1, 2, \dots\}$ *is the initial marking.*
- $L : T \to \Lambda$ *is a labeling function where* Λ *is a set of operation names.*

C.4.1 SPN and GSPN Example

Consider the motivating example again. A customer checks the proposal of $FirstBroker$ and either accepts or reject the proposal. Since the customer chooses one behavior between two choices, we represent this process with choice operator of SPA, '+', as follows:

$$choice\_decision := (accept + reject). \tag{C.2}$$

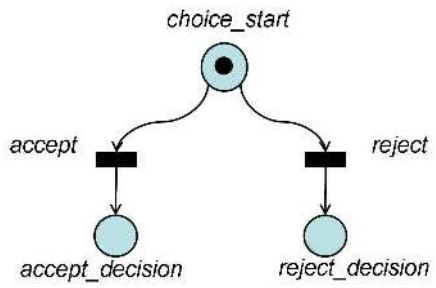


Figure C.3: The process of the choice decision

Similarly, we can map choice_decision into GSPN model as shown in Figure C.3. Here, the place, choice_start marked with a token enables both accept and reject transition. If the accept transition is fired, the token switches places from choice_start to accept_decision. On the contrary, if the reject transition is fired, the token switches places from choice_start to reject_decision.

Definition C.4.3 (MISQ Model)**.** *A MISQ Model is an 8-tuple* $(DSequence, Agent, Protocol, DState, DClass, DActivity, SPA, GSPN)$ *where:*

- *DSequence is a sequence diagram with objects, behaviors, and messages between objects.*
- *Agent is a set of objects in DSequence. We denote each element of Agent as* $a(i)$ *with* i *being the position of the element (i.e., if* $|Agent| = n$*,* $a(1)$ *and* $a(n)$ *are the leftmost and rightmost objects in DSequence).*
- *Protocol is a set of protocols. Individual protocols are sets of messages between* $a(i)$ *and* $a(j)$*, where* $i < j$ *and denoted as* $prot(i, j)$*.*
- *DState is a set of state diagrams. We denote each element of DState as* $ds(i)$ *which is the state diagram of* $a(i) \in Agent$*.*
- $DClass$ *is a set of stereo-typed class diagrams such as* $DClass - dependency$*,* $DClass - datatype$*,* $DClass - interface$*,* $DClass - protocol$*, and* $DClass - process$*.* $DClass - dependency$ *defines the dependency relationship between each element in Agent.* $DC - datatype$ *defines message contents and data classes as well as the relationship between message contents and data classes.*

$DC - interface$ defines operations. $DC - protocol$ defines the roles of the corresponding port type. $DC - process$ defines internal variables and its ports, which are connected to each element in Agent.

- *DActivity defines activity diagrams for an element in Agent.*

Next, we present several transformation procedures from one model to another in MISQ.

1. $a(i) \in Agent(i > 1)$ has communication with the left and right objects, that is, $prot(i-1, i) \neq \emptyset$ and $prot(i, i+1) \neq \emptyset$. For example, $a(1)$ has $prot(1, 2) \neq \emptyset$, and $a(n)$ has $prot(n-1, n) \neq \emptyset$.

2. For $prot(i, j), |i - j| \leq 1$. That is, each object communicates only with its immediate neighbors.

3. $|Agent| \geq 2$. That is, there are at least two objects.

Now, we present three transformation procedures: (1) UML to SPA, (2) SPA to GSPN, and (3) UML to Implementation.

C.4.1.1 Procedure 1

In this procedure, the given UML is re-captured into SPA model. It has two steps.

1. *Building Atomic processes*

   (a) Prepare $DSequence, Agent, Protocol, DState$.

   (b) Create $APset = \{x | x \in SPA\} = \emptyset$.

   (c) Set $i = 1$ and choose an $a(i) \in Agent$.

   (d) Create an atomic process, $p(i) \in SPA$.

   (e) Start transforming $ds(i)$ into $p(i)$. Transitions of $ds(i)$ are transformed to either delayed or immediate actions. If a transition does not have any temporal information, it becomes immediate action 'a'. Otherwise, λ is added and becomes the delayed action (a, λ).

   (f) If any action branch exists, it is expressed by a non-deterministic choice, '+'.

(g) A sequence of transitions in $ds(i)$ corresponds to the sequence of actions in $p(i)$.

(h) $APset = \{p(i)\} \cup APset$.

(i) If $|APset| = |Agent|$ and all $ds(i) \in DState$ is transformed, the procedure stops. Otherwise, increase i by one and go to Step 1.3.

2. *Building a Composite process*

(a) Create a process, $System \in SPA$ and $System := p(1)$. Increase i to two.

(b) Choose $p(i) \in APset$.

(c) $System = System \|_S P(i) \backslash S$, where $S \equiv prot(i-1, i)$.

(d) If $i = |APset|$ and all the $p(i)$ get combined into $System$, the procedure stops. Otherwise, increase i by one and go to Step 2.2.

C.4.1.2 Procedure 2

In this procedure, the SPA model is transformed into Petri-Net based GSPN graphical model for easier manipulation. As shown in Figure C.4, It is generally known that any SPA model can be represented as a GSPN model, and details of such translations can be found [58] [34]. In our proposal, the approach introduced in the previous research [93] is used. We do not describe the entire procedure.

C.4.1.3 Procedure 3

Once the high-level UML design has been optimized in the GSPN model, web-service implementation can finally be generated in this procedure. We use the methods described in [6], but we can use another implementation-specific method for this procedure (e.g., from UML to CORBA).

1. Based on the optimized system specification obtained in Procedure 2, $DClass-dependency, DClass-datatype, DClass-interface, DClass-protocol, DClass-process, and DActivity$ are drawn.

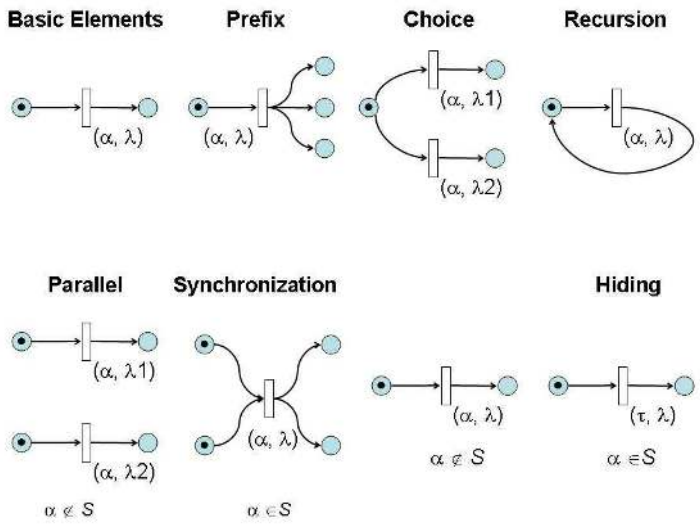


Figure C.4: Mapping SPA operations into GSPN processes

2. $DClass-dependency$ maps to an XML namespace import in WSDL. $DClass-datatype$ maps to message types and data types in WSDL. $DClass-interface$ maps to operations types in WSDL. $DClass-protocol$ maps to port and service link types in WSDL. $DClass-process$ and $DActivity$ map to process definitions in BPEL.

C.5 Illustrative Example

In this section, we demonstrate how to optimize web-service composition using the MISQ methodology. The following list summarizes notations used in this example.

- C: Customer, C's inter-arrival time follows $exp(\lambda)$.
- B: Brokerage web service.
- WS: A set of auto loan web services, $ws_j \in WS$ where $1 \leq j \leq n$. We assume that $1 \leq n \leq 4$.
- $Rate(ws)$: A loan rate returned from $ws \in WS$, $uniform(5, 6)$ is followed.
- t_o: Time-out until which B waits for $Rate(ws)$.

- $WS(S)$: A set of web services, $WS(S) \subset WS$, that successfully send a loan rate before t_o.
- $WS(F)$: A set of web services, $WS(F) \subset WS$, that fail to send a loan rate before t_o.
- $Min(Rate)$: Smallest $Rate(wsj)$, $\forall ws_j \in WS(S)$.
- $Fee(ws_j)$: Service fee that B pays to $ws_j \in WS(S)$.
- $Fee(B)$: Service fee that C pays to B.
- AR: Accept rate, $AR = \exp^{-\delta(Min(Rate)-5)}$ - $2^{(Fee(B)-10)}/2^{10}$, where δ is a preference parameter.
- PT: Profitable throughput $PT = |C| \times AR$.

PT exponentially decrease as $Min(Rate)$ increases, meaning that customers will not accept the offer if the rate is high. PT also decrease in proportion of $2^{(Fee(B)-10)}/2^{10}$ as $Fee(B)$ increases, suggesting that customers will not accept the offer if the service charge to B is high. AR expresses C's purchasing intention, whose parameters could be selected based on real market surveys. Here, however, we simply use parameters, *exp* and 2, in the interest of time.

C.5.1 Scenario

Consider the following scenario:

1. C seeks for an auto loan with a minimum interest rate, and sends an inquiry to B (C has no direct access to WS).
2. B relays C's request to each $ws_j \in WS$.
3. ws_j calculates and returns its $Rate(ws_j)$ to B.
4. The communication between B and ws_j is asynchronous with the time-out, t_o. After t_o, B does not wait for $Rate(ws_j)$ anymore. B must pay $Fee(ws_j)$ to successful ws_j, who returns $Rate(ws_j)$ within t_o.

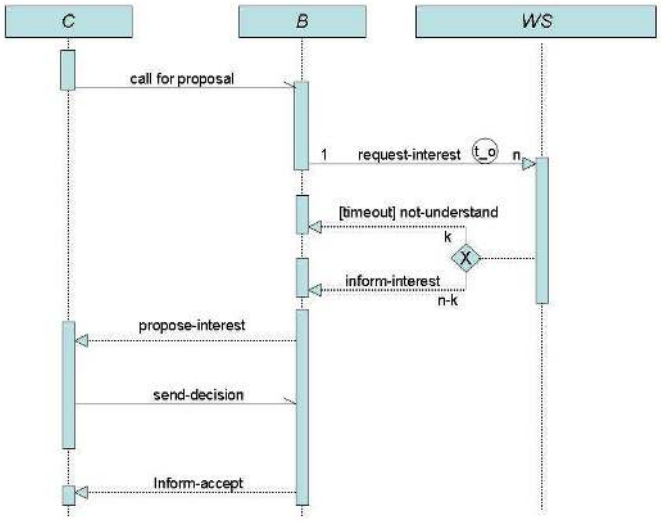


Figure C.5: Sequence diagram of the example

5. B sends $Min(Rate)$ to C.

6. If C accepts $Min(Rate)$, C pays $Fee(B)$ to B. Otherwise B cannot charge $Fee(B)$ on C.

Figure C.5 illustrates the sequence diagram of the scenario.

C.5.2 Applying MISQ to the example

We want to "maximize" the expected profit of B, who is a business adopter in the context of BSN's. Thus, the objective function, Z, representing the expected profit of B can be:

- $Z = Fee(B) \times (PT) - Fee(ws) \times |WS(S)| \times T$.

If $Z \geq 0$, B makes a profit. Z is directly proportional to PT. If $|WS|$ increases, PT is likely to increase because C has a better chance to obtain lower $Min(Rate)$, while B has to pay more fees for an increased $|WS(S)|$. Meanwhile, if $Fee(B)$ decreases, PT may increase since a low service charge can attract more C to accept the offer, making B's profit decrease. Note that there are two trade-off relations necessary to find the optimal values as follows:

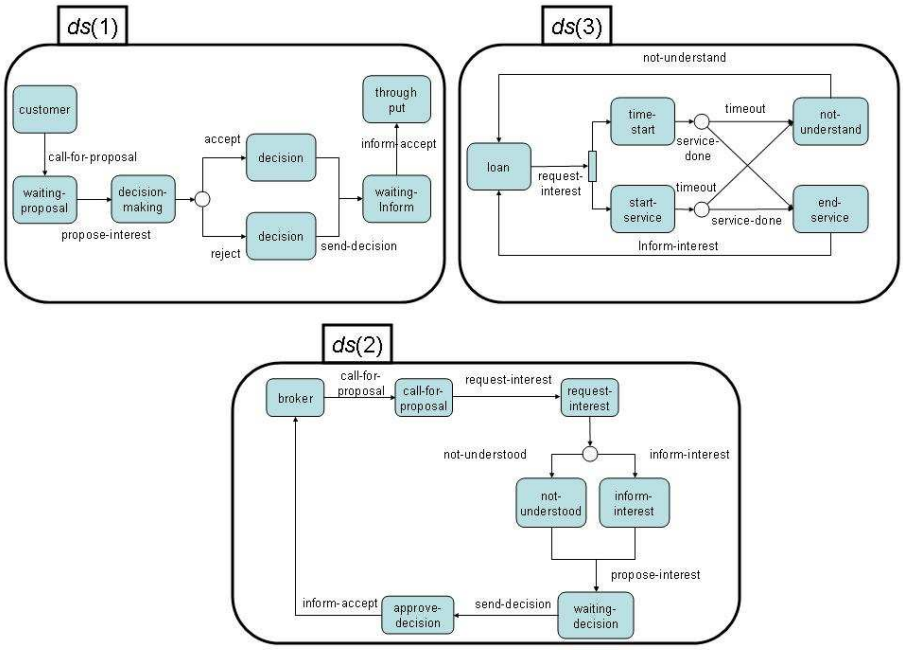


Figure C.6: Sequence diagram of the example

- $|WS| = n$: How many web services of loan companies are economical for B to use?
- $Fee(B)$: How much service charge for customers is appropriate?

Since we assumed $1 \leq n \leq 4$, we apply MISQ analysis starting with $n = 1$ and can repeat the analyses by increasing n by 1. If $n = 1$, $Agent = \{a(1), a(2), a(3)\}$, where $a(1)$ is C, $a(2)$ is B and $a(3)$ is each $ws_j \in WS$. Similarly, $DSequence = \{ds(1), ds(2), ds(3)\}$ where $ds(1)$, $ds(2)$ and $ds(3)$, and is shown in Figure C.6. $Protocol = \{prot(1,2), prot(2,3)\}$ where $prot(1,2) = \{callforproposal, propose-interest, send{-}decision, inform{-}accept\}$ and $prot(2,3) = \{request{-}interest, not{-}understand, inform-interest\}$.

C.5.3 Building atomic and composition processes.

We can first build the following atomic processes:

- customer:= call-for-proposal; propose-interest; (accept + reject); send-decision; inform-accept; throughput.

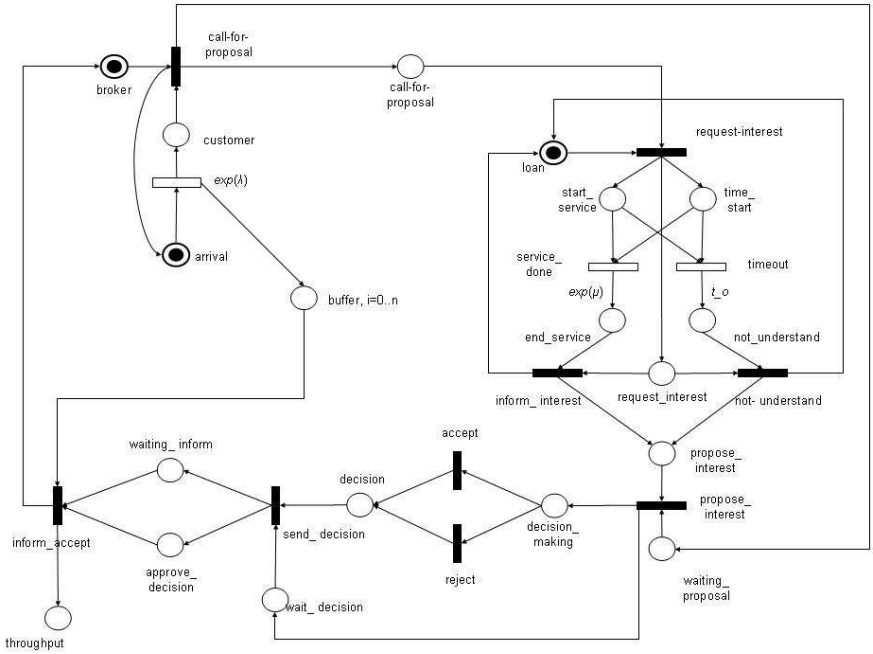


Figure C.7: Sequence diagram of the example

- broker:= call-for-proposal; request-interest; (not-understand + inform-interest); propose-interest; send-decision; inform-accept; broker.
- loan:= request-interest; ((timeout, t_o);not-understood + (service-done, μ);inform-interest); loan.

In addition to the original atomic processes, we can add two more processes, arrival and buffer, for collecting analysis data as follows:

- arrival := (gen, λ); call-for-proposal; arrival.
- buffer(i):= (gen, λ); buffer(i+1) + inform-accept; buffer(i-1), where $i \geq 1$

Next, based on the aforementioned atomic processes, we build the composite process of $System$ as follows:

- $System' := customer\|_{prot(1,2)} broker\|_{prot(2,3)} loan \backslash (prot(1,2) \cup prot(2,3))$.
- $System := (System'\|_{S1} arrival)\|_{S2} buffer \backslash (S1 \cup S2)$, where $S1$ = call-for-proposal, $S2$ = gen , propose-interest.

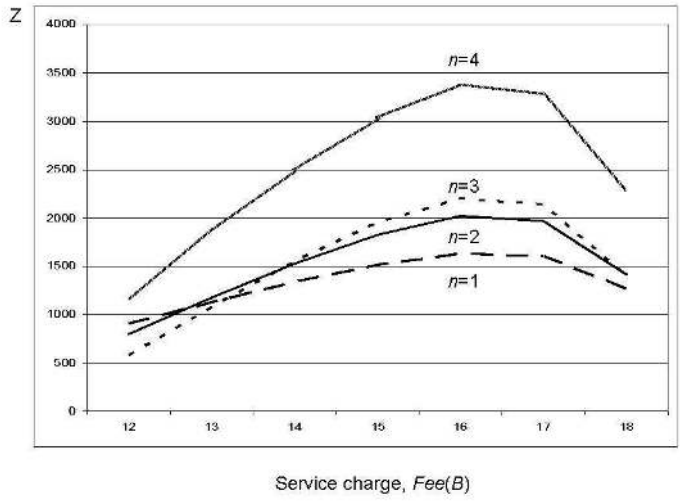


Figure C.8: Profit change according to $|WS|$ and $Fee(B)$

C.5.4 Transforming SPA into GSPN

Through the SPA to GSPN procedure, the composite process System in SPA is transformed into GSPN, as shown in Figure C.7.

C.5.5 Simulation of GSPN

We conducted simulations for four experimental cases: $|WS| = 1$, 2, 3, and 4. We assumed that $(1/\lambda) = (1/\mu) = (t_o) = 4$ hours, $\delta = 5$, and $Fee(ws_j) = \$1$. GSPN model simulation was done using HPSim [9] and the result analysis was conducted with MS Visual Basic and Excel. Simulation time was set to the same as B's life cycle, 10,000 hours. As shown in Figure C.8, the optimal setting for the scenario occurs when $|WS| = 4$, $Fee(B) = \$16$ with the expected profit of B being \$3,373.

C.5.6 High fidelity UML and Implementation

Once we acquire optimal parameters for the auto-loan example, we can build DClass-dependency as in Figure C.9. Similarly, we also can generate $DClass - datatype$, $DClass - interface$, and $DClass - protocol$ as in Figure C.10. Those models map into a WSDL file. Furthermore, we can also build $DClass - process$

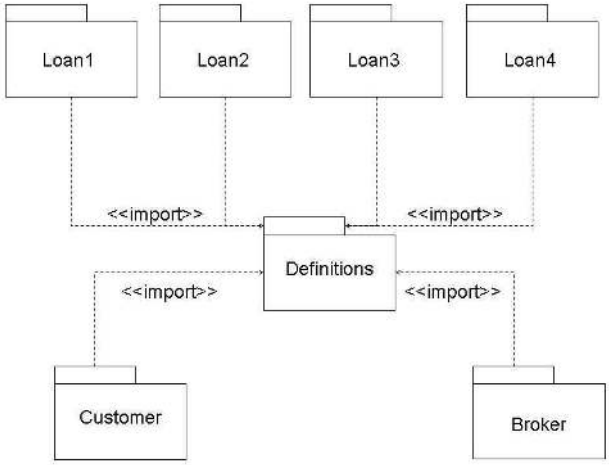


Figure C.9: Dependency Diagram of the example

in Figure C.11, and *DActvity* in Figures C.12, C.13, and C.14. These models map into a BPEL file. Some parts of implementation codes of WSDL and BPEL are illustrated in Figures C.15 and C.16, respectively.

Figure C.16 illustrates the BPEL of the example which imports the WSDL and orchestrates web services, including customer and four loan web services. The main body of the BPEL is <process> which can be divided into two parts, such as the process type definition and the process activity definition. As shown in Figures C.15 and C.16, WSDL has the important role concerning BPEL, in which the process is established based on the service model defined by WSDL. In WSDL, the two key concepts, process and partner, are modeled as WSDL services. A BPEL process reuses the definition of WSDL and can be deployed in different ways and in different scenarios. For instance, as shown in Figure C.16, the loan broker as well as four loan companies can reuse the same WSDL file, but they use it in different scenarios. The BPEL process model has a limitation in that it can conduct peer-to-peer interaction between services described in WSDL. For this peer-to-peer interaction, as shown in Figure C.15, WSDL model defines messages and portTypes. Within its portTypes, the interactions among web services are defined as operations where the corresponding messages are used as arguments [100]. Like a flow chart, BPEL provides two types of primitives: unit and control primitives. For

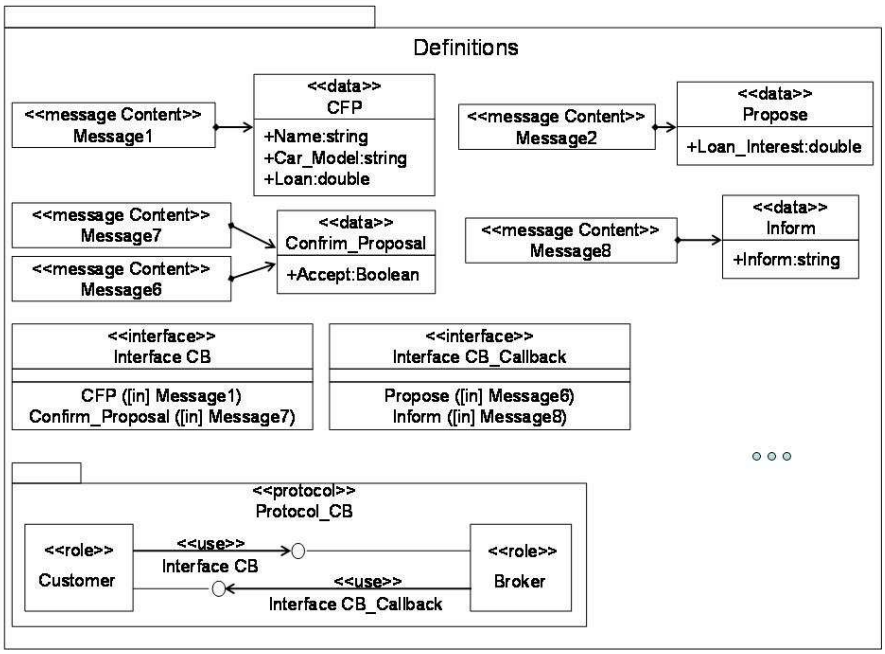


Figure C.10: Definition package of the example

the unit primitives, as shown in C.16, BPEL uses <invoke>, <receive>, <reply>, <wait> and combines them to make more complex process units. For the control primitives, it uses structural activities such as <flow>, <sequence>, <switch>, <pick> and so on. In BPEL, there is partnerLinkType which characterizes the services, which the corresponding business process communicates with. BPEL allows for maintaining data for later use during the interaction phase in the business process. A process definition is made of an activity, a series of partners and containers with specific correlation sets, and the definition of fault handlers and compensation handlers [27]. Interaction implemented in BPEL is specified based on the message exchange between web services. It is necessary that these messages are predefined in WSDL, where operations also must be defined if they will use those messages [100].

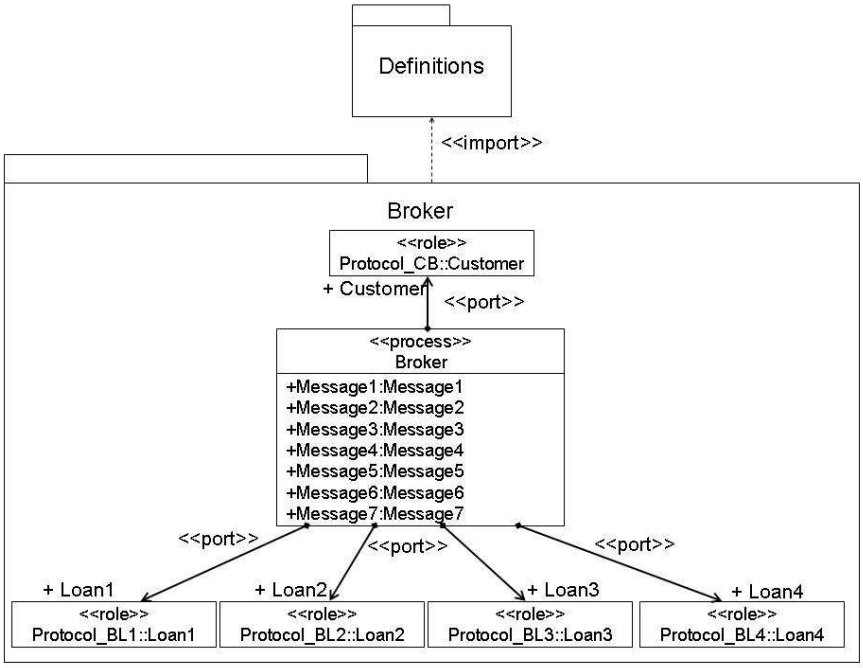


Figure C.11: Broker package of the example

C.6 Conclusion

The MISQ systematically optimizes web-service composition to identify the optimal values such as the number of ideal web services, maximum throughput, etc. There are several future research directions. In addition to simple value optimization, more functional analyses (e.g., deadlock detection or security flaw detection) can be greatly benefited from MISQ. Also, considering real-time IT provisioning and adoption enabled by BSN's, more "dynamic" optimization is a challenging goal. For instance, optimizing the dynamic workflow [57] of web-service components can greatly benefit both software vendors and business adopters. Toward this scenario, discovering, dynamically composing, and optimizing large-scale (e.g., in the range of 1,000 - 10,000) web services is a challenging problem. We approached the problem by viewing web-service composition as an AI planning problem in the main text. What has been presented in this research is thus complementary to the graph search based web-service composition research. In the near future, we plan to combine the ideas of the main text and that of MISQ to accomplish truly dynamic web-service composition methodology.

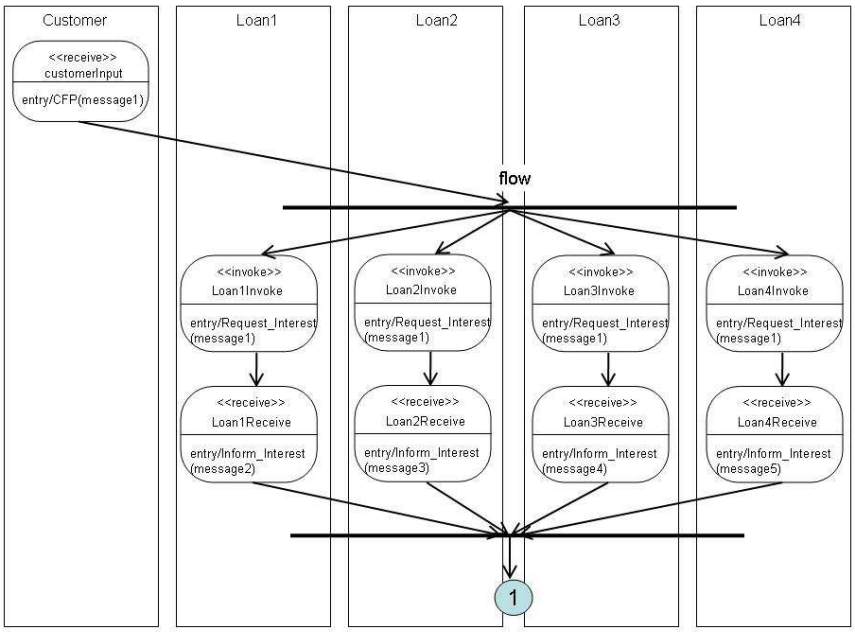


Figure C.12: Activity diagrams of the example (1)

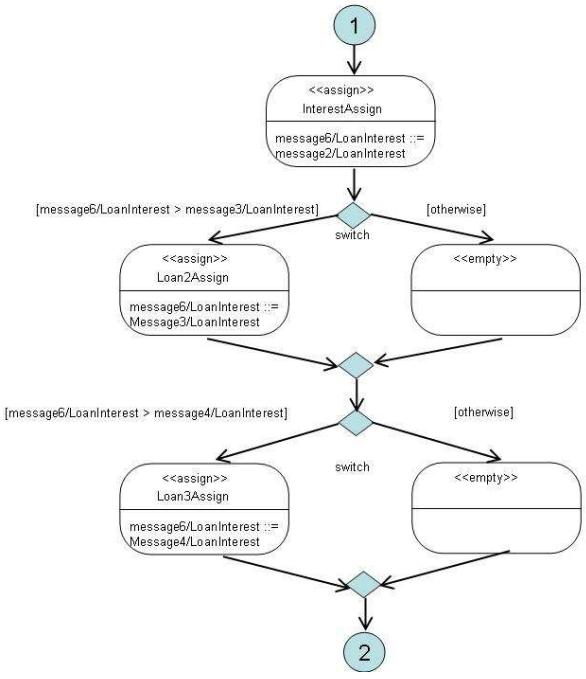


Figure C.13: Activity diagrams of the example (2)

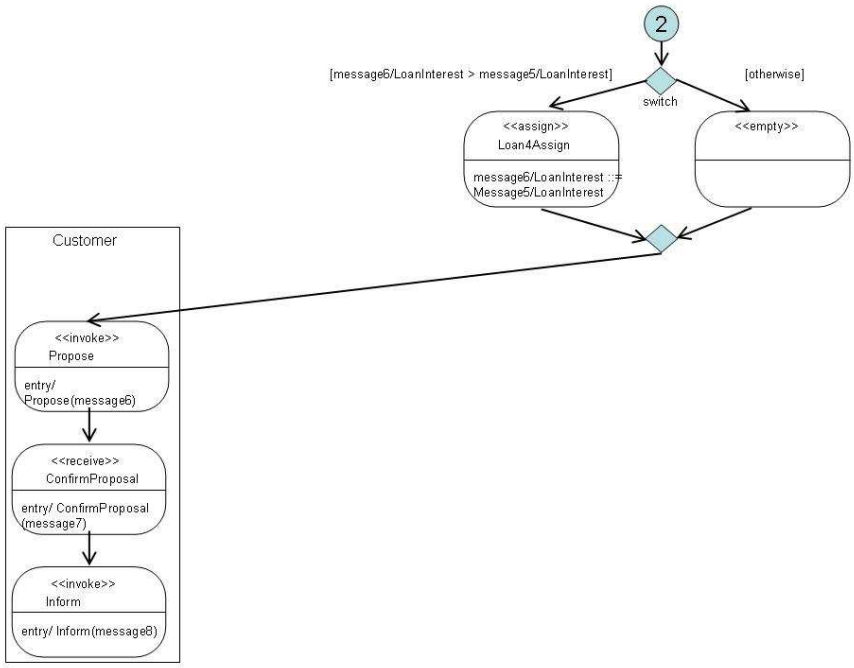


Figure C.14: Activity diagrams of the example (3)

```
<?xml version=‘‘1.0’’?> <definitions name=‘‘Broker’’ ... > <types>
 <element name = ‘‘CFP’’>
    <seqeunce>
        <element name=‘‘Name’’ type=‘‘string’’>
        <element name=‘‘Car_Model’’ type=‘‘string’’>
        <element name=‘‘Loan’’ type=‘‘int’’>
    </sequence>
 <element>
... </types> <message name=‘‘Message1’’>
    <part name=‘‘parameters’’ element=‘‘CFP’’/>
</message> ... <portType name=‘‘Interface CB’’>
    <operation name=‘‘CFP’’>
            <input message=‘‘Message1’’/>
    </operation>
    <operation name=‘‘Confirm_Proposal’’>
            <input message=‘‘Message7’’/>
    </operation>
</portType> ... <serviceLinkType name=‘‘Protocol_CB’’>
    <role name=‘‘Customer’’>
        <portType name=‘‘Interface CB_Callback’’/>
    </role>
    <role name=‘‘Loan’’>
        <portType name=‘‘Interface CB’’/>
    </role>
</serviceLinkType> ... </definitions>
```

Figure C.15: WSDL of the example

```
<process name =‘‘Broker’’ ...>
 <partners name=‘‘Customer’’ serviceLinkType=‘‘Protocol_CB’’
    partnerRole=‘‘Protocol_CB:Customer’’
    myRole=‘‘Protocoal_CB:Broker’’/>
 </partners>
  ...
 <receive name=‘‘customerInput’’ partnerLink=‘‘Customer’’
 portType=‘‘Interface CB’’ operation=‘‘CFP’’ variable=‘‘Message1’’ ... />
 <flow>
 <sequence>
           <invoke name=‘‘Loan1Invoke’’
           partnerLink=‘‘Loan1’’portType=‘‘Interface BL1’’
           operation=‘‘CFP’’ variable=‘‘Message1’’ ... />
           <recevie name=‘‘Loan1Invoke’’ partnerLink=‘‘Loan1’’
           portType=‘‘InterfaceBL1 Callback’’ operation=‘‘Propose’’
           variable=‘‘Message3’’ ... />
 </sequence>
  ...
 </flow>
 <assign name=‘‘InterestAssign’’ >
           <copy>
           <from variable=‘‘message2’’ portion=‘‘LoanInterest’’ />
           <to variable=‘‘message6’’ portion=‘‘LoanInterest’’ >
           <copy/>
 </assign>
 <switch>
           <case condition=‘‘message6/LoanInterest >
           message3/LoanInterest’’>
                 <assign name=‘‘Loan2Assign’’ >
                   <copy>
                   <from variable=‘‘message6’’
                   portion=‘‘LoanInterest’’ />
                   <to variable=‘‘message3’’
                   portion=‘‘LoanInterest’’ >
                   <copy/>
           </assign>
  <otherwise>
           <empty />
 </switch>
 ...
 <invoke name=‘‘Propose’’ partnerLink=‘‘Customer’’
 portType=‘‘Interface CB callback ’’ operation=‘‘Propose ’’
 variable=‘‘Message6’’ ... />
 <receive name=‘‘ConfirmProposal’’ partnerLink=‘‘Customer’’
 portType=‘‘Interface CB’’ operation=‘‘ConfirmProposal ’’
 variable=‘‘Message7’’ ... />
 <invoke name=‘‘Inform’’ partnerLink=‘‘Customer’’ portType=‘‘Interface CB
 callback’’ operation=‘‘Inform ’’ variable=‘‘Message8’’ ... />
</process>
```

Figure C.16: BPEL of the Broker

Appendix D

WSPR Manual

D.1 Motivation

As the emergence of service-oriented architecture provides a major boost for e-commerce agility, the number of available web services is rapidly increasing. However, when there are a large number of web services available and no single web service satisfies the given request, one has to "compose" multiple web services to fulfill the goal. Concerning this problem, we have developed and implemented an AI planning-based web-service composition algorithm titled WSPR. The WSPR outperforms state-of-the-art AI planners, such as GraphPlan and Blackbox, in composing large-scale web services (in the range of 1,000 to 50,000) based on different network topologies, and won the first runner-up award in ICEBE05 web-service contest.

D.2 How to use WSPR

D.2.1 Installation

We encourage possible users to contact us through the WSPR official homepage [126] if they want to obtain WSPR package. In addition, the full version of the technical report is downloadable from the WSPR official homepage. WSPR package is written in Python (>= Version 2.3). This package consists of three python files ("WSHSP.py," "Path.py," and "WebServicePath.py") and they must reside

in same directory.

To see the available options, simply enter:
>python WSHSP.py -h
or
>python WSHSP.py –help

D.2.2 Sample Test Set

In this manual, we assume that users can locate and download sample test sets and request sets from the ICEBE05 official homepage [54]. For detailed information about the ICEBE05 sets, please read the instructions available at the homepage.

D.2.3 Basic Usage

We will assume that Python is located in the users path, and that the user is running our application in the same directory as "WSHSP.py" is installed. The ICEBE05 challenge offers two routines, "CompositionRoutine" or "DiscoveryRoutine". Our application can distinguish the challenge routines automatically by checking the different initiation XML tag in the test request files (i.e., either "CompositionRoutine" or "DiscoveryRoutine").

- A basic example of running WSHSP.py is:
  >python WSHSP.py [options]
- A more typical example of running WSHSP.py is:
  >python WSHSP.py -i ICEBE_sample_data/out_discovery
  -g ICEBE_sample_data/discovery_config.xml.out.xml -a wsp -o result.xml

where

1. "python WSHSP.py" runs our application.
2. The " -i ICEBE_sample_data/out_discovery" says that "ICEBE_sample_data/out_discovery" is the directory to find WSDL files to use for the discovery or composition.

3. The " -g ICEBE_sample_data/discovery_config.xml.out.xml " says what file WSPR reads to define the request including initial parameters and output parameters.

4. The " -o result.xml" says that the results of WSPR will be written in "result.xml" conforming to the ICEBE05 challenge's out format.

D.3 Available Options

1. Input (-i or –input): This specifies the directory that contains WSDL files for testing.

2. Goal (-g or –goal): This specifies the "goal.xml" conforming to the ICEBE05 challenge's format that contains input and output parameters.

3. Algorithm (-a or –algo): Currently we have only "one" algorithm implemented, but plan to implement different algorithms depending on the applications. Thus, users must type "wsp" option to use WSPR algorithm. For instance,
   >python WSHSP.py -a wsp

4. Output (-o or –output): This outputs the discovery or composition results in a xml file conforimg to the ICEBE05 challenge's solution xml format. Note that the users need to specify the output file name.

5. All together: For instance, when users want to: (1) use WSDL files in "test" directory; (2) define their request in the "goal.xml" file; (3) find the composition (or discovery) using wsp algorithm; and (4) generate "result.xml" that reports the results, the following will carry out these tasks:
   >python WSHSP.py -i test -g goal.xml -a wsp -o result.xml

D.3.1 GUI version

Along with the console version above, WSPR is also provided in a GUI-version, as shown in Figure D.1. The widgets in the GUI, such as buttons and check buttons play the same roles as the command options in the previous console version.

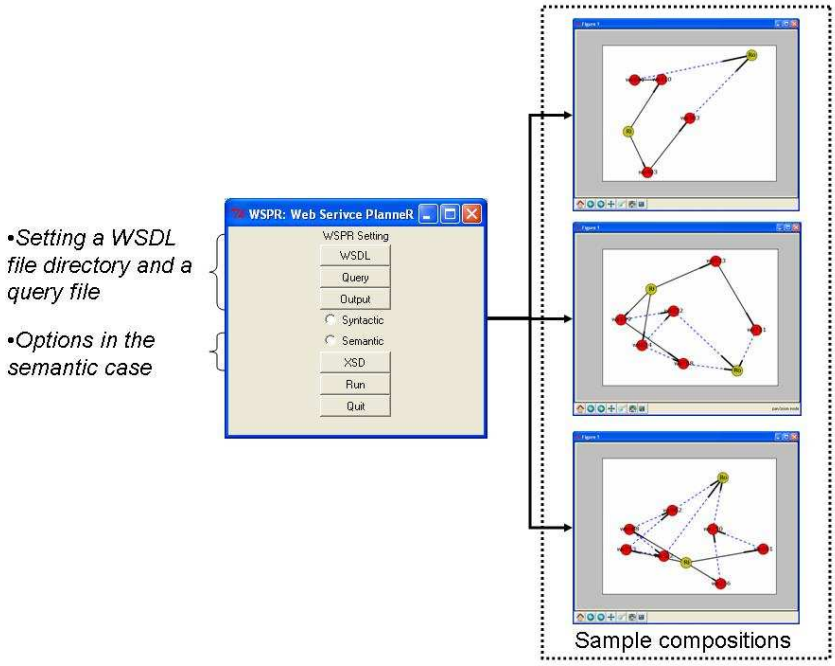


Figure D.1: GUI version of WSPR

The sample composed services are shown in the right side of Figure E.1. In the graph, each composed solution has nodes such as "r_i" and "r_o", which represent the initial condition and goal state, respectively. The directed arcs indicate the invocation flow, where a solid edge means full-matching invocation and a dotted edge represents partial-matching invocation.

D.4 Trouble Shooting

If WSPR does not show any message or if it returns errors, make sure Python was installed correctly and the executable was added to your Path system environment variable. If everything looks alright with your Path, make sure you are in the directory of the WSHSP.py file. If it is any other problem please contact us through the WSPR official homepage

Appendix E

WSBen Manual

E.1 Motivation

The database and AI community increasingly devotes attention to discovery and composition problem of large volumes of web services. As many prototypes of dedicated web-service discovery and composition tools are available, there is a strong need for a framework to analyze the capabilities and performance of such tools as early as possible. Thus, WSBen is designed and developed with an aim at providing flexible benchmarks that helps users and developers to gain insights into the characteristics of their algorithms and products.

E.2 Purpose of WSBen

WSBen is a web-service benchmark tool which produces scaled WSDL documents with user-defined characteristics and sample queries to run a user's proposal. Both documents of WSDL and sample queries are according to the DTD specified in the web-service challenges (EEE05 and ICEBE05). In addition, WSBen allows users to translate WSDL documents into a PDDL or Strips file so that they can compare their tools with existing AI planners concurrently.

E.3 How to use WSBen

E.3.1 Installation

- WSBen requires that Python version 2.3 or greater be installed.
- WSBen requires NetworkX[1] that is a Python package for the creation, manipulation, as well as the study of the structure, dynamics, and functions of complex networks.
- WSBen is downloadable from the official WSBen homepage [125].

E.3.2 Options

WSBen provides two input parameter frameworks: xTS and yTS. They are different from each other in terms of their approaches to specify the parameter cluster network. In this manual, we assume that users choose to use xTS because of its easiness to use. WSBen comes with a number of options to influence the output behavior. Users are first recommended to understand our framework to generate web services described in the full version of technical reports which are available in our official WSBen homepage.

- -j $\langle factor \rangle$: The number of clusters (jars).
- -t $\langle factor \rangle$: The (total) number of web services to produce.
- -r $\langle factor \rangle$: The parameter condense rate. This value is used to specify the size of cluster.
- -m $\langle factor1, factor2, factor3, factor4 \rangle$: The graph model of the parameter cluster network. factor1 must be one among "ba", "nws", and "er". If Factor1 is set to "ba", then Factor2 and Factor3 are assigned for N and m_0 of the Barabasi-Albert scale-free network model, respectively; Factor4 is not required. If Factor1 is set to "nws", then Factor2, Factor3, and Factor4 are assigned for N, k, and p of the Newman-Watts-Strogatz small-world network

[1] https://networkx.lanl.gov/

model, respectively. If Factor1="er", then Factor2 and Factor 3 are assigned for N and p of the Erdos-Renyi random graph model, respectively.

- -o $\langle out\_name \rangle$: It is used to specify the name of the output directory in which the generated WSDL files are located.
- -q: It generates a query file. Default name is "$\langle out\_name \rangle$.query".
- -a: It translates WSDL files into a PDDL and a Strips file. Default names are "$\langle out\_name \rangle$.pddl" and "$\langle out\_name \rangle$.strips".
- -p: It generates a report where a parameter and its usage frequency are described comma-separately. Default name is "$\langle out_name \rangle$_parDegree.txt". This file can be read by MS-Excel as an cvs file format.

E.3.3 Basic usage (examples)

In the following examples, WSBen will generate three test sets based on different parameter cluster networks as follows:

- Scale-free network:
  >python WSBen.py -j 100 -t 1000 -r 1 -m ba,100,6 -o $./ba/ - q - a - p$
- Small-world network:
  >python WSBen.py -j 100 -t 1000 -r 1 -m nws,100,6,0.1 -o $./nws/ - q - a - p$
- Random graph:
  >python WSBen.py -j 100 -t 1000 -r 1 -m er,100,0.06 -o $./er/$ -q -a -p

Each test set illustrated above can be downloaded from the WSBen official homepage. Further documentations, such as the full version of technical reports and the presentation file of WSBen system, are downloadable from the WSben official homepage too.

E.3.4 GUI version

WSBen is also provided in a GUI-version, as shown in Figure E.1. The widgets in the GUI, such as buttons and check buttons play the same roles as the command

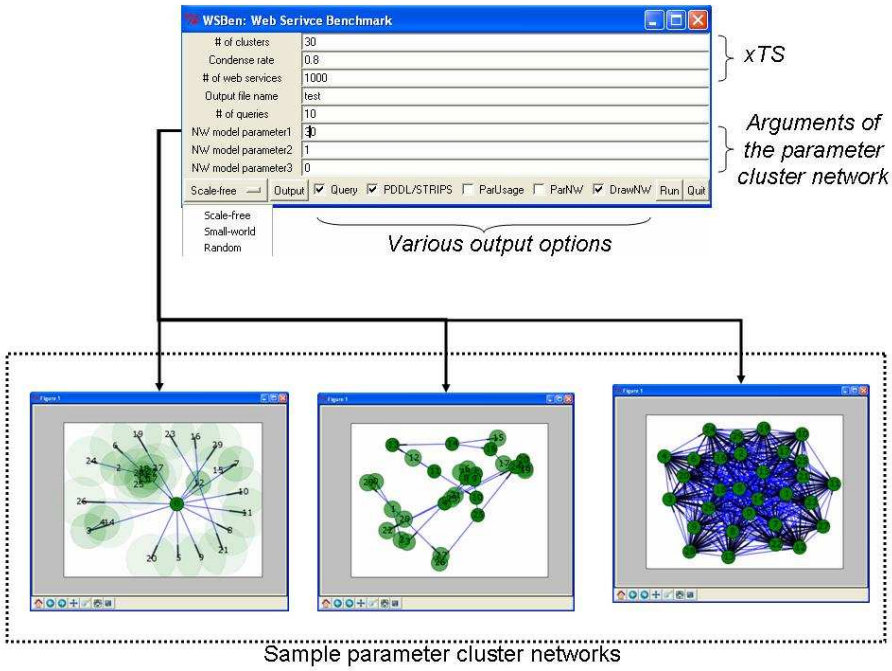


Figure E.1: GUI version of WSBen

options in the console version above. One added function in the GUI version is to visualize the parameter cluster network formed by setting xTS by values. In Figure E.1, the three networks below the GUI are such sample parameter cluster networks, where each circular node represents a cluster and edges with heads denote the web service template, from which web services are instanced. The size of node is proportional to the number of parameters in the node, while the transparency level of a node's color is inversely proportional to the degree of the node. For example, the hub cluster in a parameter cluster network that is characterized by the high degree and small number of parameters is presented by a small circle with less transparent color in the graph.

E.4 Trouble Shooting

If WSBen does not show any message or if it returns errors, make sure Python was installed correctly and the executable was added to your Path system environment variable. If everything looks alright with your Path, make sure you are in the

directory of the WSBen.py file. Any other problem please contact us through the WSBen official homepage.

Bibliography

[1] T.B. Achacoso and W.S. Yamamoto. *"AYs neuroanatomy of C. elegans for computation"*. "CRC Press, Boca Raton, FL, USA", 1992.

[2] R. Albert and A. Barabasi. "Statistical mechanics of complex networks". *Reviews of Modern Physics*, 74(1):47–95, 2002.

[3] R. Albert and A.-L. Barabasi. "Topology of evolving networks". *Phys. Rev. Lett.*, 85:5234–5237, 2000.

[4] R. Albert, H. Jeong, and A.-L. Barabasi. "The diameter of the world wide web". *Nature*, 401:130–131, 1999.

[5] I. Altintas, E. Jaeger, K. Lin, B. Ludaescher, and A. Menon. "A web service composition and deployment framework for scientific workflows". In *Proc. of the second IEEE International Conference on Web Services (ICWS)*, pages 814–815, San Diego, CA, USA, 2004.

[6] J. Amsden, T. Gardner, C. Griffin, and S. Iyengar. "UML 1.4 profile for automated business processes with a mapping to BPEL 1.0". Technical report, IBM, 2004. http://www-128.ibm.com/developerworks/rational/library/4593.html (Last accessed October 5, 2006).

[7] T. Andrews. "Business process execution language for web services, version 1.1". Technical report, OASIS, 2003. http://www106.ibm.com/developerworks/library/ws-bpel/ (Last accessed October 5, 2006).

[8] A. Ankolekar, M. Burstein, J.R. Hobbs, O. Lassila, D. Martin, D. McDermott, S.A. McIlaraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. "DAML-S: web service description for the semantic web". In *Proc. International Semantic Web Conference*, pages 348–363, Sardinia, Italy, 2002.

[9] H. Anschuetz. "HPSim Copyright © 1999-2001", 1999. http://www.winpesim.de/petrinet/e/hpsim_e.htm (Last accessed May 1, 2006).

[10] S. Arai, Y. Murakami, Y. Sugimoto, and T. Ishida. "Semantic web service architecture using multi-agent scenario description". *Lecture Notes in Artificial Intelligence*, 2891:98–109, 2003.

[11] A.-L. Barabasi and R. Albert. "Emergence of scaling in random networks". *Science*, 286(15):509–512, 1999.

[12] A. Barrett, K. Golden, and S.D. Weld. "UCPOP User's Manual". Technical Report TR 93-09-06, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA, USA, 1993.

[13] B. Benatallah, M. Dumas, Q.Z. Sheng, and A.H. Ngu. "Declarative composition and peer-to-peer provisioning of dynamic web services". pages 297–308, San Jose, CA, USA, 2002.

[14] T. Berners-Lee. "Services and semantics: web architecture". http://www.w3.org/2001/04/30-tbl (Last accessed May 1, 2006).

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

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

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

[18] Y. Bishr. *"Semantic aspects of interoperable GIS"*. PhD thesis, Wageningen Agricultural University and ITC, the Netherlands, 1997.

[19] A. Blum and M. Furst. "Fast planning through planning graph analysis". *Artificial Intelligence*, 90:281–300, 1997.

[20] B. Bollabas. *"Random graphs"*. Academic, London, UK, 1985.

[21] B. Bonet and H. Geffner. "Planning as heuristic search". *Artificial Intelligence*, 129(1-2):5–33, 2001.

[22] A. Bouguettaya, B. Benatallah, and A. Elmagarmid. *"Interconnecting heterogeneous information systems", In: A. Elmagarmid (Editor), Advances in Database Systems*. Kluwer:Norwell, MA, USA, 1998.

[23] P.A. Buhler and J.M. Vidal. "Adaptive workflow = web services + agents". In *Proc. of the first IEEE International Conference on Web Services (ICWS)*, pages 131–137, Las Vegas, NV, USA, 2003.

[24] T. Bylander. "The computational complexity of propositional STRIPS planning". *Artificial Intelligence*, 69(1-2):165–204, 1994.

[25] M. Cantera. "IT professional services forecast and trends for web services". Technical Report ITES-WW-MT-0116, Gartner Inc., 2004.

[26] J. Cardoso and A. Sheth. "Semantic e-workflow composition". *Journal of Intelligent Information Systems*, 21(3):191–225, 2003.

[27] J. Cardoso and A. Sheth. "Semantic e-workflow composition". *Journal of Intelligent Information Systems (JIIS)*, 21(3):191–225, 2004.

[28] W. W. Cohen, P. Ravikumar, and S. Fienberg. "A comparison of string distance metrics for naming-matching tasks". In *Proc. of Workshop on Information Integration on the Web (IIWEB)*, pages 73–78, Acapulco, Mexico, 2003.

[29] Corba. http://www.corba.org/ (Last accessed May 1, 2006).

[30] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *"Introduction to algorithms"*. MIT Press and McGraw-Hill, MA, USA, 2001.

[31] J. Delgado. "Emergence of social conventions in complex networks". *Artificial Intelligence*, 141:171–185, 2002.

[32] P.J. Denning. "Network Laws". *Communications of the ACM*, 47(11):15–20, 2004.

[33] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. *"Ontology matching: a machine learning approach" In: S. Staab and R. Studer (Editors), Handbook on Ontologies in Information Systems.* Springer-Velag, Berlin, Germany, 2003.

[34] S. Donatelli, H. Hermanns, J. Hillston, and M. Ribaudo. *"GSPN and SPA compared in practice: quantitative modelling in paralle systems"*. Springer, Berlin, Germany, 1995.

[35] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. "Similarity search for web services". In *Proc. of the 30th Very Large Data Bases (VLDB)*, pages 372–383, Toronto, Ontario, Canada, 2004.

[36] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma. "Dynamic workflow composition: using markov decision processes". *Int'l Journal. of Web Services Research*, 2(2):1–17, 2005.

[37] EEE05. http://www.comp.hkbu.edu.hk/ eee05/contest/ (Last accessed March 1, 2005).

[38] P. Erdos, R. Graham, and J. Nesetril. *"The mathematics of Paul Erdos"*. Springer-Verlag, Berlin, Germany, 1996.

[39] P. Erdos and A. Renyi. "On random graphs". *Publicationes Mathematicae*, 6:290–297, 1959.

[40] V. Ermolayev, N. Keberle, S. Plaksin, O. Konoenko, and V. Terziyan. "Towards a framework for agent-enabled semantic web service composition". *Int'l Journal. of Web Services Research*, 1(3):63–87, 2004.

[41] J. Fan and S. Kambhampati. "A snapshot of public web services". *SIGMOD Record*, 34(1):24–32, 2005.

[42] Ernest Friedman-Hill. "Java based expert system (JESS)". http://www.jessrules.com/jess/index.shtml (Last accessed May 1, 2006).

[43] K. Furst and T. Schmidt. "Turbulent markets need flexible supply chain communication". *Production Planning and Control*, 12(5):525–533, 2001.

[44] M. Ghallab, D. Nau, and P. Traverso. *"Automated planning: theory and practice"*. Morgan Kaufmann, San Mateo, CA, USA, 2004.

[45] S. Ghandeharizadeh, C.A. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J.L. Ambite, M. Cai, C.-C. Chen, P. Pol, R. Schmidt, and Saihong. "Proteus: a system for dynamically composing and intelligently executing web services". In *Proc. of the first IEEE International Conference on Web Services (ICWS)*, pages 17–21, Las Vegas, NV, USA, 2003.

[46] B. Gilles. "Learning in FOL with similarity measure". In *Proc. of the tenth American Association for Artificial Intelligence conference*, pages 82–87, San-Jose, CA, USA, 1992.

[47] A. Ginsberg. "An unified approach to Automatic Indexing and Information Retrieval". *IEEE Expert*, 8(5):46–56, 1993.

[48] G.V. Glass and K.T. Hopkins. *"Statistical method in education and psychology, 3/E"*. Allyn and Bacon, London, UK, 1996.

[49] V. Haarslev and R. Moller. "Description of the RACER system and its applications". In *Proc. of International Workshop on Description Logics (DL-2001)*, pages 131–141, Standford, USA, 2001.

[50] P. Haslum and H. Geffner. "Admissible heuristics for optimal planning". In *Proc. of the fifth International Conference on Artificial Intelligence Planning and Scheduling Systems (AIPS)*, pages 140–149, Breckenridge, CO, USA, 2000.

[51] J. Hoffmann and J. Koehler. "A new method to query and index sets". In *Proc. of the sixth International Joint Conference on Aritificial Intelligence(IJCAI-99)*, pages 462–467, Stockholm, Sweden, 1999.

[52] XML-RPC Homepage. http://www.xmlrpc.com/ (Last accessed May 1, 2006).

[53] E. Hovy. "Combining and standardizing large-scale practical ontologies for machine translation and other uses". In *Proc. of the First Int. Conf. on Language Resources and Evaluation(LREC)*, pages 535–542, Granada, Spain, 1998.

[54] ICEBE05. http://www.comp.hkbu.edu.hk/ ctr/wschallenge/ (Last accessed May 1, 2006).

[55] H. Kautz and B. Selman. "Unifying SAT-based and graph-based planning". In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 318–325, Stockholm, Sweden, 1999.

[56] T. Kidd. *"Agile manufacturing: forging new frontiers"*. Addison-Wesley, Reading, Massachusetts, USA, 1994.

[57] J. Kim, A. Segev, A. Patankar, and M.G. Cho. "Web services and BPEL4WS for dynamic e-business negotiation processes". pages 111–117, Las Vegas, NV, USA, 2003.

[58] K. Korenblat and C. Priami. "Extraction of π-calculus specifications from UML sequence and state diagrams". Technical Report DIT-03-007, Informatica e Telecomunicazioni, University of Trento, Italy, 2003.

[59] R. Korf. "Depth-first iterative-deeping: an optimal admissible tree search". *Artificial Intelligence*, 27(1):97–109, 1985.

[60] P. Kotinurmi. "Towards more intelligent business-to-business integration with semantic web service technologies". In *Proc. of the CIMRU-DERI-HP*, pages 33–35, 2005.

[61] B.S. Kulvatunyou, H. Cho, and Y.J. Son. "A semantic web service framework to support intelligent distributed manufacturing". *Int'l Journal of Knowledge-based and Intelligent Engineering Systems*, 9:107–127, 2005.

[62] C.T. Kwok and D.S. Weld. "Planning to gather information". In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI)*, pages 32–39, Portland, OR, USA, 1996.

[63] J. Lee, M. Kim, and Y. Lee. "Information retrieval based on conceptual distance in IS-A hierarchies". *Journal of Documentation*, 49(2):188–207, 1993.

[64] M. Levit, E. Nth, and A. Gorin. "Using EM-trained stringedit distances for approximate matching of acoustic morphemes". In *Proc. of International Conference on Spoken Language Processing (ICSLP2002)*, pages 1157–1160, Siguenza, Spain, 2002.

[65] L. Li and I. Horrocks. "A software framework for matchmaking based on Semantic Web technology". In *Proc. of the WWW 2003 conference*, pages 331–339, 2003.

[66] D. Liu, J. Peng, K.H. Law, G Wiederhold, and R.D. Sriram. "Composition of engineering web services with distributed data-flows and computations". *Advanced Engineering Informatics*, 19:25–42, 2005.

[67] A. Maedche and S. Staab. "Measuring similarity between ontologies". In *Proc. of the European Conference on Knowledge Acquisition and Management (EKAW)*, pages 251–263, Siguenza, Spain, 2002.

[68] M. Marzolla. *"Simulation-based performance modeling of UML software architectures"*. PhD thesis, Dottorato di Ricerca in Informatica, II Ciclo Nuova Serie, Dipartimento di Informatica, Universita Ca' Foscari di Venezia, Italy, 2003.

[69] D. McDermott. "A heuristic estimator for means-ends analysis in planning". In *Proc. of the third International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 142–149, Edinburgh, Scotland, 1996.

[70] D. McDermott. "The 1998 AI planning systems competition". *AI Magazine*, 21(2):35–55, 2000.

[71] S.A. McIlraith, T.C. Son, and H. Zeng. "Semantic web services". *IEEE Intelligent Systems Magazine*, 16(2):46–53, 2001.

[72] B. Medjahed, A. Bouguettaya, and A.D. Elmagarmid. "Composing web services on the semantic web". *The VLDB Journal*, 12:333–351, 2003.

[73] N. Milanovic and M. Malek. "Current solutions for web service composition". *IEEE Internet Computing*, 8(6):51–59, 2004.

[74] M. Molloy and B. Reed. "A critical point for random graphs with a given degree sequence". *Random Struct. and Algorithms*, 6:161–180, 1995.

[75] M. Molloy and B. Reed. "The size of the largest component of a random graph on a fixed degree sequence". *Combinatorics, Probability and Computing*, 7:295–306, 1998.

[76] S. Narayanan and S.A. Mcllraith. "Simulation, verification and automation composition of web services". In *Proc. of the 11th Int'l World Wide Web Conference (WWW2002)*, pages 77–88, Honolulu, USA, 2002.

[77] M. Newman, S. Strogatz, and D. Watts. "Random graph models of social networks". In *Proc. of the National Academy of Science*, pages 2566–2572, 2002.

[78] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. "Random graphs with arbitrary degree distributions and their applications". *Phys. Rev. E*, 64(026118), 2001.

[79] Nils J. Nilsson. *"Artificial Intelligence: a new synthesis"*. Morgan Kaufmann, San Francisco, CA, USA, 2001.

[80] S.-C. Oh, H. Kil, D. Lee, and S. Kumara. "WSBen: a web services discovery and composition benchmark". In *Proc. of the forth International IEEE Conference on Web Service (ICWS)*, pages 239–246, Chicago, USA, 2006.

[81] S.-C. Oh, D. Lee, and S. Kumara. "A comparative illustration of AI planning-based web services composition". *ACM SIGecom Exchanges*, 5(5):1–10, 2005.

[82] S.-C. Oh, D. Lee, and S. Kumara. "MISQ: a framework to analyze and pptimize web service composition in business service networks". *Int'l Journal. of Cases on Electronic Commerce (IJCEC)*, 1(4):35–55, 2005.

[83] S.-C. Oh, D. Lee, and S. Kumara. "WSPR: a heuristic algorithm for web service composition". *Int'l J. of Web Services Research (IJWSR)*, 4(1), 2007.

[84] S.-C. Oh, B. On, E.J. Larson, and D. Lee. "BF*: Web services discovery and composition as graph search problem". In *Proc. of the seventh International IEEE Conference on e-Technology, e-Commerce and e-Service (EEE)*, pages 784–786, Hong Kong, China, 2005.

[85] Pellet. "Pell-OWL DL Reasoner". http://www.mindswap.org/2003/pellet (Last accessed May 1, 2006).

[86] A.G. Phadke and J.S. Thorp. *"Computer relaying for power systems"*. Wiley, New York, USA, 1988.

[87] D. Pierce and B. Kuipers. "Map learning with uninterpreted sensors and effectors". *Artificial Intelligence*, 92:169–229, 1997.

[88] S.R. Ponnekanti and A. Fox. "SWORD: a developer toolkit for web service composition". In *Proc. of the 11th World Wide Web (WWW)*, pages 128–137, Honolulu, HI, USA, 2002.

[89] C. Preist, J.E. Cuadrado, S. Battle, S. Williams, and S. Grimm. "Automated business-to-business integration of a logistircs supply chain using semantic web services technology". In *Proc. of the forth International Semantic Web Conference (ISWC05)*, pages 987–1001, Galway, Ireland, 2005.

[90] PRODIGY project homepage. http://www.cs.cmu.edu//prodigy/ (Last accessed May 1, 2006).

[91] E. Rahm and P. Bernstein. "A survey of approaches to automatic schema matching". *The VLDB Journal*, 10(4):334–350, 2001.

[92] J. Rao and X. Su. "A survey of automated web service composition methods". In *Proc. of the first International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, pages 43–54, San Diego, CA, USA, 2004.

[93] M. Ribaudo. "Tackling the challenges of service composition in e-marketplaces". In *Proc. of the sixth International Workshop on Petri Nets and Performance Models*, pages 125–133, Durham, NC, USA, 1995.

[94] S.J. Russell and P. Norvig. *"Artificial Intelligence: a modern approach"*. Prentice-Hall, Englewood Cliffs, NJ, USA, 2002.

[95] S. Saadati and G. Denker. "An OWL-S editor tutorial, Version 1.1". Technical report, SRI International, Menlo Park, CA, USA, 2006. http://owlseditor.semwebcentral.org/documents/tutorial.pdf (Last accessed October 5, 2006).

[96] Salcentral. http://www.salcentral.com (Last accessed May 1, 2006).

[97] B. Selman and H.A. Kautz. "Domain-independent extension to GSAT: solving large structured satisfiability problems". In *Proc. of 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 290–295, 1993.

[98] Web service list. http://www.webservicelist.com (Last accessed May 1, 2006).

[99] G. Shegalov, M. Gillmann M, and G. Weikum. "XML-enabled workflow management for e-services across heterogeneous platforms". *The VLDB Journal*, 10(1):91–101, 2002.

[100] J. Shen, Y. Yang, and B. Lalwani. "Mapping web services specifications to process ontology: ppportunities and limitations". In *Proc. of the tenth IEEE International Workshop on Future Trends in Distribution Computing Systems (FTDCS04)*, pages 261–267, Suzhou, China, 2004.

[101] P. Shvaiko and J. Euzenat. "A survey of schema-based matching approaches". *Journal on Data Semantics*, pages 146–171, 2005.

[102] S.S. Sidiroglou, M.E. Locasto, and A.D. Keromytis. "Harware support for self-healing software services". *ACM SIGARCH Computer Architecture News*, 33(1):42–47, 2005.

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

[104] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. "Adding semantics to web services standards". In *Proc. of the first IEEE International Conference on Web Services (ICWS)*, pages 395–401, Las Vegas, NV, USA, 2003.

[105] I.J. Timm and P.-O. Woelk. "Ontology-based capability management for distributed problem solving in the manufacturing domain". *Lecture Notes in Artificial Intelligence*, 2831:168–179, 2003.

[106] E. Voorhees. *"Using WordNet for text retrieval", In: C. Fellbaum (Editor), WordNet: An Electronic Lexical Database.* MIT Press, Cambridge, MA, USA, 1998.

[107] T. Vossen, M. Ball, A. Lotem, and D. Nau. "On the use of integer programming models in AI planning". In *Proc. the 16th National Conference on Artificial Intelligence (AAAI)*, pages 304–309, Orlando, FL, USA, 1999.

[108] W3C. "Annotated DAML+OIL ontology markup". http://www.w3.org/TR/daml+oil-walkthru/ (Last accessed May 1, 2006).

[109] W3C. "OWL web ontology language". http://www.w3.org/TR/owl-features/ (Last accessed May 1, 2006).

[110] W3C. "OWL web ontology language for services (OWL-S)". http://www.w3.org/Submission/2004/07/ (Last accessed May 1, 2006).

[111] W3C. "Semantic web". http://www.w3.org/2001/sw (Last accessed May 1, 2006).

[112] W3C. "Simple Object Access Protocol (SOAP) 1.2". http://www.w3.org/TR/2003/REC-soap12-part1-20030624/ (Last accessed March 10, 2005).

[113] W3C. "UDDI 3.0 technical white paper". http://uddi.org/ (Last accessed March 10, 2005.

[114] W3C. "Web services activity (web site)". http://www.w3c.org/2002/ws/ (Last accessed March 10, 2005).

[115] W3C. "Web Services Description Language (WSDL) 2.0". http://www.w3.org/TR/2004/WD-wsdl20-20040803/ (Last accessed March 10, 2005).

[116] Y. Wang and E. Stroulia. "Semantic structure matching for assessing web service similarity". In *Proc. of the First International Conference on Service Oriented Computing*, pages 194–207, Trento, Italy, 2003.

[117] S. Wasserman and K. Faust. *"Social network analysis"*. "Cambridge University Press, Cambridge, UK", 1994.

[118] D.J. Watts. *"The dynamics of networks between order and radnomness"*. Princeton Univ. Press, Princeton, NJ, USA, 1999.

[119] D.J. Watts and S.H. Strogatz. "Collective dynamics of 'small-world' networks". *Nature*, 393(4):440–442, 1998.

[120] WebserviceX.NET. http://www.webservicex.com (Last accessed May 1, 2006).

[121] D.S. Weld. "Recent advances in AI planning". *AI Magazine*, 20(2):93–123, 1999.

[122] H. S. Wilf. *"Generatingfunctionology"*. Academic Press, London, UK, 1994.

[123] A.T. Winfree. *"The geometry of biological time"*. "Springer, New York, USA", 1980.

[124] WS-Diamond. "Web services - diagnosability, monitoring and diagnosis". http://wsdiamond.di.unito.it/ (Last accessed May 1, 2005).

[125] WSBen. http://www2.ie.psu.edu/Kumara/Research/lisq/index_files/wsben/WSBen.htm (Last accessed August 1, 2006).

[126] WSPR. http://www2.ie.psu.edu/Kumara/Research/lisq/index_files/wspr/WSPR.htm (Last accessed August 1, 2006).

[127] J. Wu and Z. Wu. "Similarity-based web service matchmaking". In *Proc. of the 2005 IEEE International Conference on Service Computing (SCC'05)*, pages 287–294, Orlando, FL, USA, 2005.

[128] xMethod. http://www.xmethod.com (Last accessed May 1, 2005).

[129] J. Yang, W. Heuvel, and M.P. Papazoglou. "Tackling the challenges of service composition in e-marketplaces". In *Proc. of IEEE Int'l Workshop on Research Issues in Data Engineering (RIDE02)*, pages 125–133, San Jose, CA, USA, 2002.

[130] J. Zhang and L.-J. Zhang. "Web services quality testing". *Int'l Journal. of Web Services Research*, 2(2):1–4, 2005.

[131] R. Zhang, B. Arpinar, and B. AlemanMeza. "Automatic composition of semantic web service". In *Proc. of the first IEEE International Conference on Web Services (ICWS)*, pages 38–41, Las Vegas, NV, USA, 2003.

Vita

Seog-Chan Oh

Seog-Chan Oh was born in Seoul, Korea on July 27, 1971. He recived his B.S. and M.S. degrees in Industrial Engineering from the Dongguk University in 1993 and 1996 respectively. During his M.S. study his major research interest was in the area of *Machine Learning*. After he finished his M.S. study he joined Daewoo Information Systems Co., Seoul, Korea. For seven years, he had mostly worked as a IT consultant in manufacturing simulation, supply chain management and product life-cycle management areas. In 2002 fall, he enrolled in the Ph.D. program in Industrial Engineering at the Pennsylvania State University. During his Ph.D. study he was employed as a research assistant in the Department of Industrial Engineering, participating in several rewarding research projects sponsored by the United State Marine Corps and the General Motor Research and Development Center, under the supervision of Dr. Soundar Kumara. His research interests include web-service based service-oriented architecture (SOA) and its applications to semantic web, supply chain, business workflow, and product life cycle. In this context, he is interested in composing distributed services in an effective and efficient way under a wide range of conditions including deterministic and stochastic environments, large-scale service networks, and complex underlying network topology. In general, his research uses methods and techniques drawn from AI planning, multi-agents, OR, graph theory, ontology, and data mining.

He has a certification of Professional Engineer at Information Management issued by the Korean government.