The Islamic University–Gaza
Research and Postgraduate Affairs
Faculty of Information Technology
Master of Information Technology

الجامعـــــــة الإســـلاميـة – غـزة
شئون البحث العلمي والدراسات العليا
كليـــة تكـنـولـوجيـا المعـلـومـات
ماجستير تكـنـولـوجيـا المعـلـومات

Semantic Web Services Composition Using Enhanced Beam Stack Search

تجميع خدمات الويب دلالياً باستخدام خوارزمية البحث مكدسة الحزمة المعززة

Teejan Tajeddean El-Khazendar

Supervised by

Dr. Rebhi Soliman Baraka

Associate Professor of Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Information Technology

June/2017

إقـــــرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

Semantic Web Services Composition Using Beam Stack Search

تجميع خدمات الويب دلالياً باستخدام خوارزمية مكدسة الحزمة

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وأن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل الاخرين لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

Declaration

I understand the nature of plagiarism, and I am aware of the University's policy on this.

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted by others elsewhere for any other degree or qualification.

Student's name:	تيجان تاج الدين الخزندار	اسم الطالب:
Signature:	تيجان الخزندار	التوقيع:
Date:	7/3/2017	التاريخ:

بسم الله الرحمن الرحيم

الجامعة الإسلامية غزة
The Islamic University of Gaza

مكتب نائب الرئيس للبحث العلمي والدراسات العليا هاتف داخلي: 1150

الرقم: ج س غ/35/ Ref:

التاريخ: 2017/07/05 Date:

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحثة/ **تيجان تاج الدين جار الله الخزندار** لنيل درجة الماجستير في **كلية *تكنولوجيا المعلومات*** **برنامج تكنولوجيا المعلومات** وموضوعها:

"تجميع خدمات الويب دلالياً باستخدام خوارزمية البحث مكدسة الحزمة المعززة"

"Semantic Web services Composition Using Enhanced Beam Stack Search"

وبعد المناقشة التي تمت اليوم الأربعاء 10 شوال 1438هـ، الموافق 2017/07/05م الساعة الثانية عشر ظهراً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

د. ربحي سليمان بركة	مشرفاً و رئيساً	.....
أ.د. علاء مصطفى الهليس	مناقشاً داخلياً	.....
د. تامر سعد فطاير	مناقشاً خارجياً	.....

وبعد المداولة أوصت اللجنة بمنح الباحثة درجة الماجستير في **كلية *تكنولوجيا المعلومات*/ برنامج تكنولوجيا المعلومات.**

واللجنة إذ تمنحه هذه الدرجة فإنها توصيها بتقوى الله ولزوم طاعته وأن يسخر علمها في خدمة دينها ووطنها.

والله ولي التوفيق،،،

نائب الرئيس لشئون البحث العلمي والدراسات العليا

أ.د. عبدالرؤوف علي المناعمة

Abstract

Semantic web services composition is a set of web services and a user request, we need to find the best applicable sequence of web services satisfying the user's request, fulfiling his requirements. Each web service has functional and non-functional requirements. The problem, is finding the web services composition that fulfills the non-functional requirements automatically without without involvement the user.

In this research, we take into consideration the non-functional properties to form a quality based web services composition specifically depending on response time and throughput (the quantity of efficiency produced over time). We enhanced the heuristic Beam Stack Search algorithm and employed it to search automatically through the web services search space for a composition satisfying the required qualities.

Number of experiments were conducted to form compositions on several web services test set sizes; 5000, 10000, and 15000, such that running the Enhanced Beam Stack Search with different beam width sizes; 120, 150, 300, and 600. The results indicate the ability of the algorithm to achieve the required compositions with the respective qualities. The beam width parameter of the algorithm plays an important role on the quality of the formed composition. As the beam width increases, the throughput of the formed composition decreases and vice versa, i.e., results obtained when using the 15000 test set size for optimal solution throughput are 848, 773, 311, and 192 sequentially by using the mentioned consequent beam widths increasingly.

Results also showed that as the test set size increases, the algorithm performs better in terms of throughput. If the user wants the minimum response time he will take the first found solution. While, if he is interested in the best-found throughput, he will choose the optimal solution.

Keywords: semantic, web services, composition, automatic, Beam Stack Search

الملخص

إن مشكلة خدمات الويب الدلالية هي عبارة مجموعه من خدمات الويب. يتطلب علينا إيجاد أفضل تسلسل مناسب من خدمات الويب يلبي طلب المستخدم و تحقق متطلباته. يوجد متطلبات وظيفيه و غير وظيفيه لكل خدمه ويب. تتمحور المشكلة حول إيجاد تجميع خدمات الويب الذي يحقق المتطلبات الغير وظيفية بشكل أوتوماتيكي دون أي تدخل من المستخدم. في هذا البحث، نأخذ بعين الاعتبار الصفات الغير وظيفية لتكوين تجميع خدمة ويب بناء على الجودة بالأخص جودة زمن الإستجابة و الكفاءة (الكفاءة المنتجة خلال فترة زمنية). قمنا بتعديل خوارزمية مكدسة الحزمة الإرشادية و توظيفها للبحث عبر مجموعة خدمات الويب عن تجميع خدمات ويب يلائم الجودة المطلوبه بشكل أوتوماتيكي. قمنا بعمل مجموعة من التجارب لتكوين تجميعات خدمة الويب باستعمال أكثر من مجموعه خدمات ويب بحثية أحجامها؛ 5000، 10000، 15000، كما قمنا باستعمال خوارزمية مكدسة الحزمة المعدلة على عرض حزم مختلفة؛ 120، 150، 300، 600. النتائج توضح إمكانية الخوارزمية للوصول إلى التجميعات المطلوبه بالجودة المطلوبة. متغير عرض الحزمة للخوارزمية يلعب دوراً هاماً حيث يُأثر على جودة تجميع خدمات الويب المُنشأ. كلما إزداد عرض الحزمة، تقل كفاءة الإنتاجية لتجميع خدمات الويب المتكون و العكس صحيح، على سبيل المثال النتائج التي حصلنا عليها عند استعمال مجموعة البحث التي حجمها 15000، أعطت كفاءة مثالية 848، 773، 311، 192 بالتسلسل مع كل عرض حزمة تصاعدياً.

كما أظهرت النتائج أنه كلما ازداد حجم مجموعة البحث، كلما كانت كفاءة الخوارزمية أفضل في إعطاء قيمة إنتاجية مثالية. في حال كان المستخدم يريد أقل زمن استجابة، سوف يأخذ أول حل تم إيجاده. أما في حال كان مهتم في أفضل كفاء، فسوف يختار الحل المثالي الذي تعطيه الخوارزمية المُعدلة.

كلمات مفتاحية: دلالي، خدمات ويب، تجميع، أوتوماتيكي، خوارزمية مكدسة الحزمة

Epigraph Page

"In the middle of difficulty lies opportunity."
Albert Einstein

Dedication

“The family is one of nature's masterpieces” -- George Santayana

This thesis is dedicated with love and affection to my family who supported me through my life journey.

Acknowledgment

First of all, I would like to thank my Master's Thesis advisor Dr. Rebhi Soliman Baraka for his support and guidance. Our continually meetings and discussions made this thesis possible.

I would also like to thank my friends for being always close to me.

Finally, thanks to my family for supporting me and inspiring me to achieve my goal.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ARA*	Anytime Reparing A*
BnB	Branch and Bound
BPEL	Business Proccess Excution Language
DCBSS	Divide and Conquer Beam Stack Search
ffsT	First Found Solution Time
ffsTH	First Found Solution Throughput
osT	Optimal Solution Time
osTh	Optimal Solution Throughput
OWL	Web Ontology Language
RWA*	Restarting Window A*
SOA	Service Oriented Architecture
THqr	Throughput Quality Ratio
Tqr	Time Quality Ratio
WSCI	Web Service Choreography Interface
WSDL	Web Service Description Language
WSLA	Web Service Level Agreement
WSMO	Web Service Modeling Ontology

Chapter 1
Introduction

Chapter 1
Introduction

In this chapter, we present an introduction to our research. The first section is dedicated for the background of our research. The statement of the problem is introduced in the second section. The focus of the third section is on the main and specific objectives of the research. The significance of the research is presented in the fourth section. The scope and limitations of the research are covered in the fifth section. In the sixth section, a brief description is given to the research methodology. An overview of the thesis is summarized in the last section.

1.1 Background and Context

Web services is a description for a set of associated functions that is available over the web through programming. Web services are loosely coupled, allows dedicated binding, also they are reusable software components. Web services have three entities that are the service requester, service provider and the registry (Medjahed, Bouguettaya, & Elmagarmid, 2003).

The procedure of combining several web services into one coarse-grained service in order to produce more composite functions is called web services composition (Oh, Lee, & Kumara, 2006). Web service composition gives a unified service that has some supplementary values.

Web services discovery is concerned with finding out the best applicable service among functionally similar services that meet the requirements of users, consequently, we must define a set of well-defined quality of services criteria and user preferences to help in the web service discovery (Seo, Jeong, & Song, 2005).

Web services composition problem, is such that there are a set of web services and a user request given, and we want to find the shortest sequence of web services satisfying the user's request. But since web services composition problem solution have to discover services that fulfil the functional and non-functional requirements including the quality of services according to the user request. Therefore, the desired

web service composition problem solution, will not be the shortest path, but the web services composition with the optimal gathered quality of services value (Bartalos & Bieliková, 2012).

Web service description language (WSDL**)** is considered as the language used to define a web service and represents the **syntactic description**. While WSDL describes the structure of the input and output, without the meaning of the data, this makes the automated web service composition challenging (Medjahed, Bouguettaya, & Elmagarmid, 2003).

A **semantic description** of web services is required for automatic discovery of these services, while current web services methods offer the syntactic description, that are difficult for the requester and the provider to understand the input and output. Semantic web services consist of both, the mixture of web services and the semantic web. With regard to the semantic web, Web Ontology Language (OWL) and Web Service Modeling Ontology (WSMO) are two techniques that can be used for service composition (Feier, et al., 2005). The use of semantic web services is to combine data and services from various sources with preserving their meaning. While discovering and combining web services, a value-added service is provided by semantic web services to complete the domain tasks (Mirbel & Crescenzo, 2010).

Web services are usually defined based on their functional parameters (input/output parameters), while the parameters of quality of service are used to describe the behavior of the service. The quality of service solves the problem of discovering the best service between the functional similar services, it makes the selection process depends on the non-functional requirements. That makes the quality of service capable of being used as the leading factor for ranking the web services. During the selection procedure, after matching the functional requirements, the web service with high quality of service value will be chosen firstly (Sivasubramanian, Ilavarasan, & Vadivelou, 2009). The web service activity sequential order flow can be expressed using several languages such as BPEL4WS (Andrews, et al., 2003) and WSCI (Arkin, et al., 2002).

Search algorithms have been used for sometime to solve problems in various fields such as large scale combination. Trying to solve the problem using the existing

search algorithms, which works on finding an optimal solution, it may take long a time in computations to complete, while this delay is not allowed due to time restrictions for the customers.

Generally, search algorithms have two significant problems when applied to large and complex problems. The first one is the problem of memory needs of the search methods especially the best first methods becomes expensive. The second one is the problem of time where search algorithms needs a lot of time to reach the best solution (Vadlamudi, Aine, & Chakrabarti, 2011).

A previous study (Shehu, Epiphaniou, & Safdar, 2014) showed a full review of the techniques that treat this search problem as NP-hard problem. It presents the concepts of quality of services aware web service composition, concentrating on quality of services properties, workflow model and quality of services aggregation functions.

Many researchers have offered different automated methods to solve the problem of semantic web services composition (McIlraith & Son, 2002), (Sheshagiri, DesJardins, & Finin, 2003), and (Wu, Parsia, Sirin, Hendler, & Nau, 2003). An important study was presented by Kil and Nam (2013) proposes using the heuristic Beam Stack Search algorithm (described in full details in Section 2.9) in solving quality of web services aware web services composition problem. This study is the only study that employed the Beam Stack Search algorithm to solve the problem of web services composition, while we also used it in our study to use the Beam Stack Search but in another way in order to perform better results.

The goal of this research is centered around enhancing an algorithm to find a specific web service among a set of web services under the order of the client meeting a specific quality criteria and to altimately be part of a composition. Enhanced Beam Stack Search algorithm enhances forming the web services composition by reconstructing each time a newly discovered solution. The Beam Stack Search frequently improves the overall solution by realizing better solutions for web services composition until finding the optimal solution. While there are two most important solutions records among the found solutions for the user, they are concentrated on finding the first fast solution and the solution with the best throughput value. At the

beginning, the Enahnced Beam Stack Search algorithm gives all the possible solutions where the user can take the first fast solution by terminating the algorithm directly after finding the first solution. Alternatively, it might complete searching for more solutions for users who are not interested in time and they can wait to find the solution with the best throughput. Then after the Enahnced Beam Stack Search algorithm finishes processing all the possible solutions, it terminates by itself, and returns the best-found path for forming the composition depending on the best-estimated response time and throughput. There are many non-functional requirements which the user may be interested in. In our research, we chose the response time since it is important to deliver the service in a good time. Also, we chose the throughput variable because it measures the quantity of efficiency produced over time, throughput is very important since there is no need for un-efficient web services.

In our research, we applied a number of experiments to form compositions by running the Enhanced Beam Stack Search using web services test set sizes; 5000, 10000, and 15000, with beam width sizes; 120, 150, 300, and 600. Conducting experiments on the different used test set sizes and diverse beam width sizes allowed us to find valuable experimental results.

1.2 Statement of the Problem

A set of web services is given by a service provider and a user request is given. The user wants to find the best web service composition that meets specific functional and non-functional requirements. Using automatic search techniques, many solutions can be formed to solve the web service composition problem fulfilling the functional requirements but the major challenge is finding a solution that also fulfils the non-functional requirements according to the user request automatically. This solution must not only consider the shortest time quality of the composition, but also the optimal throughput quality of the services forming the composition.

We implemented an Enhanced Beam Stack Search algorithm to process a set of web services with their functional and non-functional requirements, inorder to construct a set of web services compositions which are functionally similar but differs in their non-functional requirements.

1.3 Objectives

1.3.1 Main Objective

The main objective of this research is to design an algorithm based on Beam Stack search algorithm to perform semantic web services composition automatically using a set of web services to achieve the user's request, taking into consideration the solution quality. The solution quality of web services composition depends on the response time and on the throughput of the composition.

1.3.2 Specific Objectives

The specific objectives of the research are:

1. To collect and analyze a set of web services with their syntactic descriptions of the functionality they offer, and semantic descriptions utilizing their qualities.
2. To analyze Beam Stack Search algorithm to propose a suitable modification to make it applicable to solve the web service composition problem
3. To design the enhanced approach that solves the semantic web service composition problem based on Beam Stack Search
4. To implement the algorithm and conduct a number of experiments to measure the quality of the approach
5. To evaluate the algorithm. The ratio of “the throughput of the first fast solution” to “the best throughput solution”, and “the response time ratio of the first fast solution” to “the best-found solution”, are used to assess the algorithm’s efficiency.

1.4 Research Significance

This research solves the problem of web services composition problem taking into consideration user specified qualities. The user determines his request and chooses among, e.g., a first found solution or wait for the optimal solution to be discovered by the Enahnced Beam Stack Search algorithm. Enhanced Beam Stack Search discovers frequently improved solutions and realizes the optimal solution.

The Enahnced Beam Stack Search algorithm serves two types of customers. The first type is customers who need the fastest solution, and the other type is customers who care about composition solution throughput quality

The importance of the research stems from its ability to improve a general search technique such as Beam Stack Search and then employ it within the area of web services, particularly, the composition problem.

1.5 Scope and Limitations

The quality of web services has a wide range, such as security and other factors. In this research, we only focus on the factors of throughput and response time for finding each solution for the required web services by the user.

Results evaluation will be conducted in order to determine the potential advantages of using the algorithm of Beam Stack Search to solve the web service composition problem

Regarding the data sets, they are not real web services but rather experimental sets related to Acme Packet company services collected and prepared by (Blake, Weise, & Bleul, 2010) as WSDL files for the syntactic description of the services and their associated web service level agreements (WSLA) files which hold the semantics of the services including the qualities. More on these data sets can be found in Section 4.5.

1.6 Research Methodology

To achieve the objectives of the research, we follow the following methodology as shown in Figure 1.1:

Step 1. Reviewing works related to using Beam Stack Search in web services composition problem, important subjects related to the field of semantic web services and semantic web services composition problem techniques as well as related search algorithms.

Step 2. Finding and collecting the suitable web services data which will be used as the search space by the algorithm during experiments.

Step 3. Preparing and processing the collected data to be used in the experiments. While the data is stored in a web service description language (WSDL) format. We have to prepare the data as a java file to be ready for use.

Step 4. Studying the original Beam Stack Search algorithm and modify it as needed to be suitable for our purpose to search through the web services data set. depending on its original mechanism. It uses backtracking method depending on a specific beam width suitable to the used web services set size. In the set of web services, some web services are candidates of specific functionality but they differ in their non-functional requirements.

Step 5. Preparing service model which is a WSDL file containing the client`s web service requests to be searched by the algorithm to form the composition.

Step 6. Performing the required experiments using the prepared files to find the required service by the client file through the set of web services using the Enhanced Beam Stack Search algorithm.

Step 7. Study the efficiency of the approach based on the Enhanced Beam Stack Search algorithm (Zhou & Hansen, 2005).

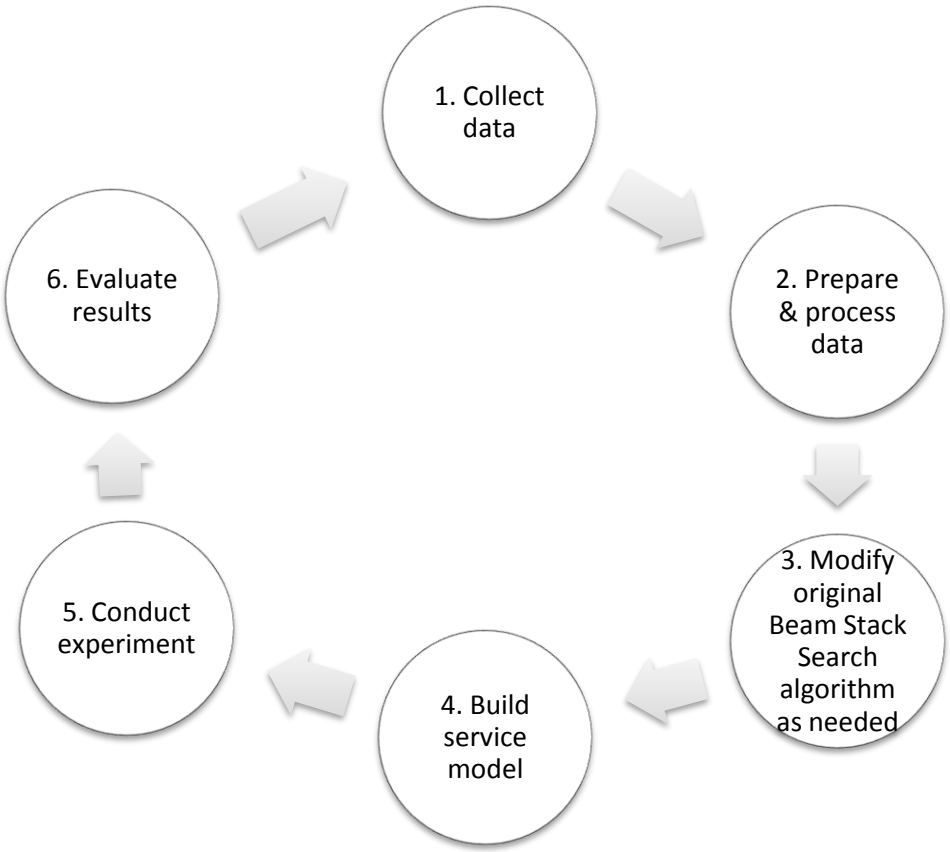


Figure (1. 1): Steps of Research Methodology

1.7 Organization of the Thesis

The thesis is organized as follows. Theoretical and technical foundations are discussed in Chapter 2. The related works are reviewed in Chapter 3. The proposed approach is described in Chapter 4. Chapter 5 is devoted to analyze and discuss the results of the approach. In Chapter 6, conclusion and recommendations are given.

Chapter 2
Theoretical and Technical Foundation

Chapter 2
Theoretical and Technical Foundation

A substantial amount of research has been done on web services composition. This chapter covers the theoretical and technical foundations related to web services, their description, discovery, selection, semantics, and composition. Heuristic search algorithms, Beam Search and Beam Stack Search and their complexities are explained and how they are used in the web services composition.

2.1 Web Services

Web services can be any application reachable to other applications through the web. This definition is open, it says that anything has an URL can be considered as a web service. For example, any reachable program over the web with a fixed application programming interfaces, and available with supplementary descriptive information on some guide can be considered as a web service (UDDI Consortium, 2001).

Web services is given by the world-wide-web consortium (W3C) (Austin et al., 2004) as "a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols". This definition stresses how web services must work, defined, described, and discovered. Web services must be not only running, but they also have to be described and advertised, so it will be possible to write clients which link and interact with them. Simply, web services are interoperable software components that can be used in application integration and component-based application development and can be integrated into supplementary complex dispersed applications (Alonso, et al., 2004).

Web service description language (WSDL) is written using XML to describe web services as endpoints set which is functioning on messages containing document(s) information or containing procedure(s) information. The processes and messages are conceptually described in the WSDL file, then engaged to a concrete

network protocol and message format inorder to outline an endpoint. Associated concrete endpoints are joint into services (abstract endpoints). WSDL is a language that is able to be extended to permit description of endpoints and their messages irrespective of what is the message formats or network protocols used to connect (Christensen, et al., 2001).

2.2 Web Services Discovery

As the demand for web services usage is increasing, various questions arise about the approaches and techniques to determine the more appropriate web service to use. Actually, there are considerable issues beyond the finding of a web service. Web services discovery mechanisms have an important role in the cooperation among business procedures and customers based on accepted web standards (Garofalakis, et al., 2006). The major subject in web services discovery is finding out the best applicable service among functionally similar services that meet the requirements of users.

Web services discovery can be considered as a match-making process (Sycara, Klusch, Widoff, & Lu, 1999) or the process of discovering a suitable service provider for a service requester over an internal proxy (Decker, Sycara, & Williamson, 1997). Generally, web services discovery starts by service suppliers when they advertise their abilities to middle brokers (registries). After that, brokers store this information, then a service client asks the brokers best matching his demanded capabilities. At the end, the broker efforts to match the client request against the stored advertisements.

Service discovery may be accomplished manually or automatically using specific mechanisms. While in both cases, the searching interface should be able to make a comparison between the supplied capabilities and the required functionality (Booth, et al., 2004).

2.3 Web Services Selection

Services from diverse providers should be selected carefully inorder to be integrated into a composite web service irrespective of their platforms, performance speeds, or even their locations inorder to carry-out complex business operations and transactions (Yu, Zhang, & Lin, 2007). The input of web services selection phase is a

set of services levels, where each level includes web services with the same functionalities, but they may differ in other non-functional features like the quality of services characteristics (Moghaddam & Davis, 2014). The customer may select the required service manually while the construction of the web service composition time depends on some extra data resources or choose the service randomly from available candidates, or may use automated web services composition techniques (Wang & Vassileva, 2007).

2.4 Web Services Composition

A Services Oriented Architecture (SOA) is a set of services connecting with each other that may contain either simple data or it could contain two or more services performing some activity (Barry & Associates, 2017). The goal of Service Oriented Architecture (SOA) is to offer a loosely-coupled combination or/and composition of web services existing in diverse systems and programmed using various programming languages. Commonly, web services are platform independent applications which can be invoked through the internet. Easing the gathering of web services to form composite web services, is a significant functionality in SOA (HU & Wang, 2008).

Generally, web services composition problem is represented by, that we are given a set of web services and a user request and we want to find the shortest sequence of web services fulfilling the user request. The problem of automatically gathering web services inorder to form compositions that enhance given user priorities is often denoted as the automated web service composition problem (Doshi, Vembu, & Zhao, 2011). Web services composition needs to find service suppliers that fulfil functional and non-functional requirements, which takes the quality of web services constraints in consideration.

2.5 Web Services Composition Classifications

Web services composition can be categorized depending on three significant specifications which depend on automation degree of the composition, the complexity of the composition and the scale of the composition (Albreshne & Pasquier, 2010) and (Oh, Lee, & Kumara, 2006).

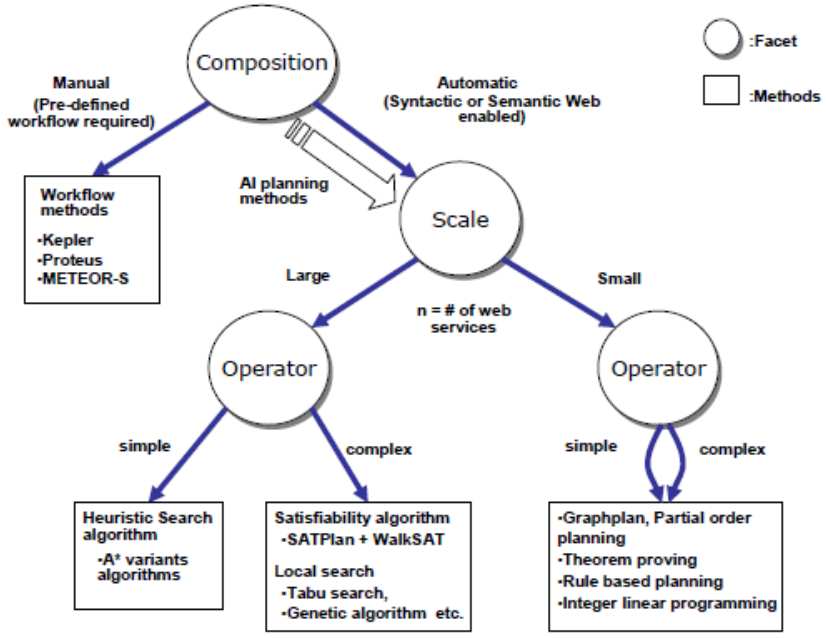


Figure (2. 1): A decision tree of artificial intelligence solutions for the web service composition problem (Oh, Lee, & Kumara, 2006).

Figure 2.1 illustrates the web services composition as a decision tree of artificial intelligence solutions classifications in a simple flowchart. We identify briefly these classifications as follows:

2.5.1. Composition Automation Degree

Composition can be manual, **automatic** or **semi-automatic**. A manual composition has to be performed by domain experts because it is zero automated, so it relies on the user experience. While automatic composition is performed using software programs so ordinary users can use it.

- Manual composition approach: is the traditional approach where users must be familiar with the domain, they choose suitable web services and include them into a coherent workflow. Users might depend on a GUI based software to make the composition easy, even though, it requires expertise and is susceptible to errors. Processes are defined by a process execution language like BPEL. Many existing tools have plug-ins for enabling manual composition such as Net-Beans (NetBeans.org, 2016), and JOpera (JOpera, 2016). This approach is not easy to be used because it requests a lot of knowledge by the user and it comes to be more and more challenging with the explosion of web

services resources, so it is not suitable for large-scale web service composition problems.

- Automatic composition approach: we work in this path. This approach works without user participation, it is used when the user has a set of restrictions and priorities and he has no method pattern. It depends on discovering services for performing abstract processes which are defined previously. The automatic tools, try to find the available web services that semantically correspond as much as possible to the user's requirements.
- Semi-automatic composition approach: which is also called the interactive composition approach, in this type of composition, the system often supports users to discover, filter and combine automatically the wanted services through matching the user's requests for the existing services. Furthermore, it allows end users to be involved all the time throughout the composition process.

2.5.2. **Composition Operators**

Web services composition can be performed using either **simple** or **complex operators**. Simple operators web service composition searches using a sequence of AND operators. For example, "web service a_2, AND web service b_6, AND web service c_9, AND ..." also, it does not contain any restrictions. Complex operators web service composition use additional operators (such as OR, XOR and NOT operators) or restrictions (for example, request r prefers web services located in Europe to those located in Asia).

2.5.3. **Composition Scale**

There is **small** and **large** scale web services composition. Exhaustive search algorithms could only work for small scale web service composition problem. Large scale problems, estimated algorithms which find sub-optimal solutions are preferred (Sivasubramanian, Ilavarasan, & Vadivelou, 2009).

Various methods can be used to solve the web services composition problem automatically such as heuristic search algorithms, linear programming, and Genetic algorithm.

Based on the artificial intelligence methods shown in Figure 2.1 and thinking about web services composition as a large scale scenario, we need to use simple search operators through the composition process with the various available candidates. This is why we are using a heuristic algorithm without thinking about Genetic methods. Specifically, the used heuristic technique in this research to solve the web services composition problem is the Beam Stack Search algorithm.

In Section 2.7, we present various concepts related to heuristic search algorithms, Beam Search, Beam Search, Beam Stack Search and using them in the composition of web services.

2.6 BPEL as a Web Services Composition Language

Performing web services composition can be done through BPEL orchestration. Orchestration is the technique that is used to combine web services, while the concerned web services are restrained and controlled by a single endpoint essential process which is simply another web service. Web services can be combined without being aware that they are playing a part in a larger business process (Albreshne, Fuhrer, & Pasquier, 2009).

Business Process Execution Language for Web Services (BPEL, WS-BPEL, BPEL4WS) which is commonly referred to by BPEL, is the new standard for outlining business procedures with services composition. It is the foundation stone of Service Oriented Architecture (SOA).

A BPEL process flow expresses the order in which the involved web services in a composition are composed, either in sequence or in parallel.

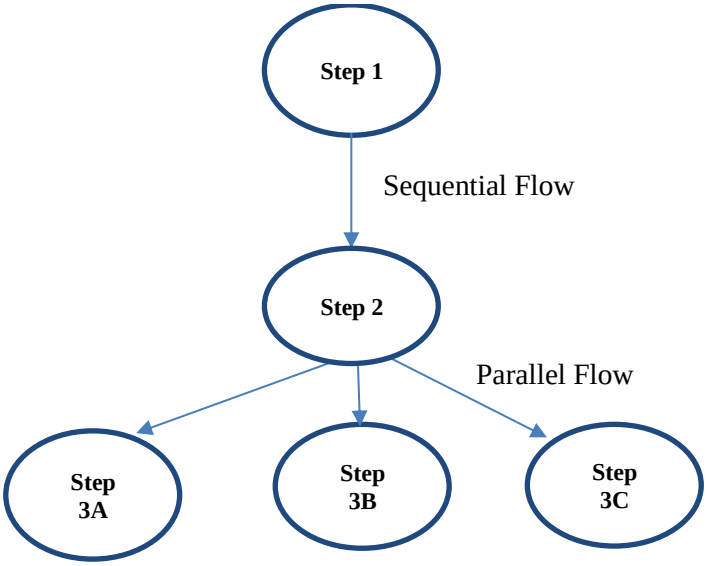


Figure (2. 2): BPEL process flow

Figure 2.2 illustrates how BPEL process flow in its two flow types, Sequential flow, and the Parallel flow type.

A BPEL process consists of a set of actions. It interacts with exterior associate services through a WSDL mediator. A BPEL process, defines the execution order, conditional behaviours, and activities. Additionally, it defines the namespace, ports, operations, partner link types, and messages that are needed to determine the process actions. WSDL files are required in order to generate an effective, executable BPEL definition (Albreshne, Fuhrer, & Pasquier, 2009).

2.7 Heuristic Search Algorithms

Heuristic search algorithms solve optimizing problems through finding a regular fast suboptimal solution, then working on finding enhanced solutions when given additional time. For a fast solution, anytime search algorithms are characteristically greedy with respect to the heuristic cost h. There are various heuristic search algorithms (referred to an anytime A* method) as we review some of them.

Likhachev, Gordon, and Thrun (2003) Anytime Repairing A* (ARA*) algorithm adopts an allowable heuristic and minimizes the weight w each time, to be used in the cost function as follows:

$$f(n) = g(n) + w * h(n)\ , \qquad w > 1 \qquad (3.1)$$

ARA* works by executing A* several times, starting with a large w and scaling down w value before each execution until w = 1. Consequently, after each individual search, a solution is ensured to be by a factor w of finest solution which assign the denotation of optimality for the solution. ARA* algorithm, proves w acceptability of the present solution. This indirectly prunes the search space like that no state has ever expanded whose f'-value is bigger than the value of the present solution. When decreasing the value of w, ARA* changes the correspondig f'-values of all the states in "Open" set approbate to the new weight. Additionally, ARA* eschews reexpanding states through search round, while each round is the part of search among two weight changes. Each time a shorter path to a specific state is found, and that state has previously been expanded in the present search round, the state is not expanded again directly. Alternatively it is stored in a separate list, which will be put in the "Open"set only at the launch of the following round. The logical basis beyond this, is that even without reexpanding states, the subsequent found solution is definite to be in the current sub optimality bound (ARA* algorithm). As this method concentrates on finding sub-optimal solution, the composition of web services does not benefit from it.

The heuristic Beam Stack Search algorithm (Zhou & Hansen, 2005), is based on breadth-first search. In Beam Stack Search algorithm, just the maximum talented nodes in each level of the search space are expanded, where the beam width w is given by the user. The algorithm recollects which nodes have not until now been expanded and gets back to them in a subsequent time. However, beam-stack search discovers the whole search space beneath the selected states before it backtracks on its resolution. We can say that Beam Stack Search makes the Beam Search into a complete search algorithm by applying a backtracking mechanism, simply it iterates the beam algorithm to find all the candidate solutions. Discussed widely in Section 2.9.

Additional iterative anytime heuristic search algorithm called anytime Window A* algorithm (Aine, Chakrabarti, & Kumar, 2007) that is also based on breadth-first search like the beamstack search do. In this method, the expansion of each node is restricted through a "sliding window" involving levels of the search graph, which

means that the sliding window moves downwards in a depth first manner, each time a state in a higher level than the earlier expanded one ,the window slides down to that level of the exploration space. Only states in that level and the h levels above can be expanded, while h is the height of the sliding window. Initially h=0, and it increases by one every time a new solution is found. This algorithm can suffer from its strong depth first concentration if the heuristic approximations are inexact and vary significantly far away.

Another heuristic search algorithm is called “The Joy of Forgetting: Faster Anytime Search via Restarting” (Richter, Thayer, & Ruml, 2010), a suitable name for the proper technique since it works on initiating the search from the initial node whenever a new solution found.

The searchers advice to restart Window A* algorithm. Therefore, this algorithm is referred to as Restarting Window A* algorithm (RWA*). Simply, we can describe RWA* algorithm as it iteratively runs the Window A* algorithm with reducing weight, constantly reexpanding states once it finds a cheaper path.

RWA* differs from ARA*algorithm and Anytime Window A* algorithm, that it does not preserve the “Open” list between phases. Each time an improved solution is created,the search empties the “Open” list and start over from the initial state. This algorithm as unusual addes another third list to the ordinary “Open” and “Closed” lists found in the previous algorithms, which is called “Seen”. When a new search phase starts, the states from the old “Closed” list are moved to the “Seen” list. This algorithm will behave typically such as the Window A* algorithm if a generated state in the new search has never been generated before bymeans that it does not belong to any list (neither “Open” nor “Closed” nor “Seen”), which means that RWA* then will calculate the heuristic value of the state and insert it into the “Open” list. Also, it will behave again as the Window A* if the state has been came across before in this search phase (it is either in “Open” list or “Closed” list). Then RWA* will reinsert the state into the “Open” list only if it found a shorter path to the goal state. While there is a thired case for reached state, which that this state has been came across in previous search phases but not in the present phase bymeans that it belongs to “Seen” list. Then RWA* will have another behavior, it will find the heuristic value of this state from the

phase which it was previousely came by across, rather than calculating heuristic value again. Also, the RWA* algorithm examinations the previously found path to the state is cheaper or it found a new better path inorder to keep the better one. Finaly it moves the state from "Seen" list to the "Open" list. We can conclude that this algorithm prevents calculating the heuristic value of a state more than one time, and previous effort (Window A* algorthim) is used in making usage of the best path to a goal state found. However RWA* algorithm's restarts gives additional flexibility in finding different solutions, but it may reexpand many states that were previously expanded in earlier phases which will waste memory and time

Through this section we can conclude that, Beam Stack Search algorithm prevents from states rexpandings which preserve time and memory space. Also, it calculates the heuristic values for states just one time through the algorithm which makes us think more better about depending on in our project to solve the web services composition problem. Depending on Beam Stack Search we can find a fast sub-optimal solution wich will be the first found solution, and whenever the user has more time the search algorithm will keep on going until finding the best solution.

2.8 Beam Search

Beam Search is considered as a modification of branch-and-bound (BnB) search. It uses an inadmissible pruning rule. The Beam Search selects only the most promising nodes for more branching at each level of the search graph using heuristic, while the remaining nodes are pruned forever. Beam denotes the nodes that will be explored in each level and the beam width w denotes the size of search in the beam (number of nodes to be explored). Beam search expands nodes in breadth-first order and uses a fixed beam width, Beam Search method is alike best-first search mechanism (Wikimedia Foundation, Beam search, 2017)

2.9 Beam Stack Search

Although the Beam Search algorithm can find a prompt solution, it may bypass the optimal solution. This is due to the method of the Beam Search. In this method only picked points are examined in each level.

Zhou and Hansen (Zhou & Hansen, 2005) developed an algorithm named Beam Stack Search in order to optimize the Beam Search algorithm using divide and conquer technique. In the Beam Stack Search algorithm, the optimal solution is found by reiterating the Beam Search.

Shown in Figure 2.4, when the Beam Stack Search finishes one iteration of the Beam Search, it archives the search advancement and goes on to the following iteration to catch an enhanced solution. To follow the nodes which have been called, the Beam Stack Search processes the beam stack which encompasses an element for every level. The element of the beam stack determines the range of the cost [f_{min} ; f_{max}) so that only successor nodes having cost in this range are saved in the next level. The algorithm rejects any successor nodes with a cost less than the lower bound f_{min} or greater than or equal to the upper bound f_{max}, when expanding nodes in a level related to an element in the range [f_{min} ; f_{max}). The element of level zero having one start node is saved at the bottom of the beam stack. However, the element related to the presently expanding level is saved at the top of the beam stack (line 18 in **Function Search** shown in Figure 2.5). When the algorithm make its first expansion for a node in a level, it sets the first element of the equivalent level to have the range [0,U) (line 4 in **Algorithm DCBSS)**, where U is the present upper bound of the cost. If the level size becomes larger than the beam width (line 19 in **Function Search**), then the Beam Stack Search do an inadmissible pruning for nodes with the maximum cost (line 21 to 27 in **Function Search**) to save space for new nodes.

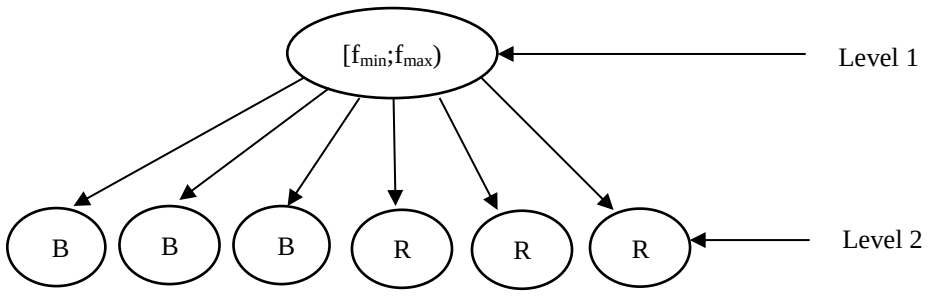


Figure (2. 3): A tree for illustrating levels and expanded nodes.

In Figure 2.3, it is assumed that the beam width is equal to three, and Level 2 holds the expanded nodes from Level 1. Each pruned node in Figure 3 has a cost greater than the cost of the other nodes that have B sign inside, then the three nodes with smaller costs are not pruned.

At the moment that the search algorithm prunes nodes in a level, it varies the f_{max} of the element in the previous level to have the value of the minimum cost of the pruned nodes. This guarantees that the search algorithm will not produce any successor node having a cost exceeds or equals to the minimum cost of the pruned nodes which indeed became the Upper cost.

When search algorithm arrives a level all its successor nodes have a cost larger than U, we name it an empty level.

When search algorithm backtracks, it deletes from the top of the beam stack successive elements with an f_{max} larger than or equal to U (line 12 and 13 in **Algorithm DCBSS**).

Search algorithm back-tracks the level linked to the element on the top of the beam stack stand for the lowest level which comprises specific node(s) having one or more pruned successors.

Each time the search algorithm backtracks to a level, the Beam Search is obliged to allow a variant set of successor nodes by modifying the range of the element [$f_{\min}$; f_{max}) saved in the beam stack element linked with the level. When the algorithm backtracks to a level, the new $f_{\min}$ will have the value of the present f_{max} (line 18 in **Algorithm DCBSS**) and the new f_{max} will have the value of the upper bound U (line 19 in **Algorithm DCBSS**). This means that the range [$f_{\min}$, f_{max}) is shifted to [f_{max}, U). The search will not stop until all the nodes in the present level are expanded.

To create new successor nodes that might have been inadmissibly pruned in the prior visit of the level, the expansion of the nodes that were expanded in the latest visit of this level should be repeated.

Once the f_{max} of beam stack element of the level is larger than or equal to the upper bound when all nodes in the level have been expanded, it can be said that the backtracking of the level is complete. This implies that successor nodes with cost

within the range [f_{min},U) has not been pruned meanwhile the pervious time the search algorithm backtracked to the level. Consequently, successor nodes with cost in the range [0;U) should have been created for the level.

The search algorithm does not terminate immediately when it discovers a solution, it goes on to explore better solutions. The algorithm will finish when the beam stack is blank (all levels are backtracking-complete). It is simply showed that the best solution found must be optimal, then Beam Stack Search is an anytime algorithm that discovers an early solution, and goes on to find better solutions until meeting an optimal solution (line 7 and 8 in **Algorithm DCBSS**). Note that the search algorithm updates U the upper bound every time it discovers a better solution (line 9 in **Algorithm DCBSS**).

Next is the pseudocode for the Beam Stack Search algorithm as presented by Zhou and Hansen (2005).

```
1   Algorithm DCBSS: Divide and Conquer Beam Stack Search (Node start, Node
2   goal, Real U, Integer relay)
3   Beam_stack = Ø;
4   Beam_stack.push([0,U)); // initialize beam stack
5   bestPath = null; // initialize optimal solution path
6       while Beam_stack.top() ≠ null;
7           solution-path = Search(start, goal, U, relay);
8           if solution-path ≠ null then
9               best_path = solution-path;
10              U = Cost(solution-path);
11              Print (solution-path);
12      While Beam_stack.top().fmax > || = U do  // fmax upper bound of cost
13      Beam_stack.pop();
14      End while
15      If Beam_stack.isEmpty() then
16      Return bestPath;
17      Print(bestPath + " is the optimal path " );
18      Beam_stack.top().fmin= Beam_stack.top().fmax;  // fmin lower bound of cost
19      Beam_stack.top().fmax=U;
20  End while
```

Figure (2. 4): Divide and Conquer Beam Stack Search Algorithm

The above divide and conquer Beam Stack Search algorithm uses the following search algorithm (Zhou & Hansen, 2005) and iterate over it to find the solution as required.

```
1   Function Search(Node start, Node goal, Real U, Integer relay)
2   best_goal = null;
3   open[0] = {start};
4   l=0; // start level while l is the index of layer
5   open[l] = Ø;// index of level
6   closed[0] = Ø;
7       while open[l] ≠ Ø or open[l+1] ≠ Ø; do
8         while open[l] ≠ Ø; do // the current level is not empty
9           node = argmin_n{ cost(n) | n ϵ open[l] } // expand node
10          open[l] = open[l]\{node} // remove the expanded node from the open set
11          closed[l] = closed[l] U {node} // add the expanded node to the closed set
12                if (node = goal)
13                   then best_goal = node;
14                   set U = g(best_goal); /* g(node) is the cost of the best_goal path from
15                   the start node to the goal node  */
16                   End;
17                Else
18                Node.expand(beam-stack.top()) // top level workflow automatically
19                If layerSize(l+1)>w then
20                   Keep = keep the best w nodes ϵ open[l+1];
21                   Prune = {n | n ϵ open[l+1] && n ∉ Keep };
22                   Beam_stack.top().f_max = min(cost(n) | n ϵ prune };
23                   For each n ϵ Prune do
24                      open[l+1] = open[l+1] \ n
25                      delete n
26                   then Keep = open[l+1]; // after the pruning
27                End
28      End while;
29      if 1 < l ≤ relay or l > relay + 1 then
30        for each n∈ Closed[l−1] do /* delete previous layer */
31          Closed[l−1]←Closed[l−1]\{n}
32         delete n
33        end for
34      l = l+1; // move to the next level
35      Open[l+1] = Ø
36      Closed[l] = Ø
37       Beam_stack.push([0,U)); // new item in the stack
38      untill
39          If best_goal ≠ null; then // delayed solution reconstruction
```

```
40      Return solutionReconstruction(best_goal); /* solutionReconstruction is divide-
41      and- conquer solution reconstruction technique */
42      Else
43      Return null;
44      End if
```

Figure (2. 5): Function Search Used by Beam Stack Search Algorithm

It has been hypothesized that all the successor nodes have variant costs, which permits the use of the costs of successor nodes to decide in which order to prune nodes when memory is complete. The importance of assembling nodes according to the cost is that the search algorithm discovers nodes with the minimum cost initially. This implies that the algorithm primarily discovers the best encouraging nodes. However, some nodes may have the same cost, at this case a tie breaking rule must be used to execute a whole ordering on nodes.

There are many options to manage the case in which some nodes have the same cost. The search algorithm can break ties depending on the state encoding of a node, to ensure the uniqueness of the cost. Or the Beam Stack Search may use domain-specific information. For multiple sequence alignment, an entire assembling, can be depended on the coordinate of a node in an n-dimensional hypercube (n: number of aligned sequences).

Beam Stack Search permits some unbroken ties, as long as the number of ties in a level is smaller than the beam width. It runs under bounded memory and is guaranteed to find an optimal solution, it uses open and closed sets to store all the generated nodes of a search graph in memory (**Open set** is used to store boundary search nodes, and the **Closed set** is used to store previously expanded nodes).

It is important to know that the first part of Beam Stack Search, before any backtracking is applied through the algorithm, is the same as the Beam Search; often it finds a first solution very fast, then it is an anytime algorithm that explores a sequence of enhanced solutions before reaching to optimality.

Beam Stack Search contains both breadth-first branch-and-bound BFBnB search and depth-first branch-and-bound DFBnB search as exceptional cases. As when the beam width is:

1. one, beam-stack search is equivalent to depth-first search branch-and-bound search
2. larger than or equal to the size of the largest level, beam-stack search is equivalent to breadth-first branch-and bound search, and no backtracking occurs

In the other cases, it utilizes a hybrid strategy in the search processes that combines BFBnB search and DFBnB search and offers an elastic tradeoff between existing memory and the time overhead of backtracking. Figure 2.6 gives a simple illustration for the depth-first search and the breadth-first search techniques (O'Keefe & Costa, 2015).

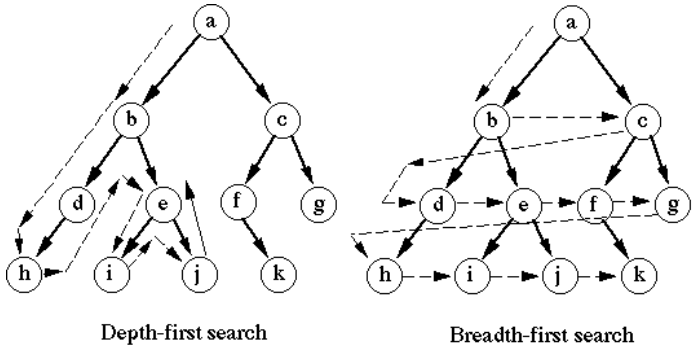


Figure (2. 6): Depth-first and breadth-first search techniques

For allowing divide-and-conquer solution reconstruction in the described algorithm, relay node technique is used. In the relay node technique, each node pasts the midpoint supplies an indicator to the start node, that is reserved in memory. For uncomplicatedness, all of the relay nodes are stored in the same layer, called the relay layer. The algorithm stores four layers; the relay layer, the presently expanding layer, its descendant layer, and its previous layer.

For each found goal node, a comparison between the costs of the current found one and the saved best solution. In case the cost of the newly found node is better that the best solution, it will be set as the best solution and its cost will be the upper cost (line 12 to 16 in Function Search).

2.10 Solving Web Services Composition Problem Using Beam Stack Search

As presented in Section 2.5, we explained why our study is using the heuristic type of search algorithms which is formalizes as the A* alternative algorithms of the artificial intelligence. Studying the Beam Stack Search, and realizing the flexibility to edit it to search through set of web services, led us to take the step in our project for discovering web services compositions depending on the Beam Stack Search algorithm. The advantage of using Beam Stack Search algorithm that it searches level by level, so web services set will be used as subsets in levels, while each level contains a subset of web services performing the same functionality, but differ in the quality (the non-functional requirements for the service), which demonstrates the meaning of web services selection (Section 2.3). The advantage of using it, that in each level the technique works on examining the top number of web services in quality under the range of the specified beam width that is given by the user. After that, it moves to the next level and apply the same method of choosing the best specific number of web services in the level, while it keeps the rest of web services from each level to be examined in another loop until it passes over all of the possible and available solutions, referring to that keeping the rest services in a stack to examine it the next loop prevent the re-expanding of nodes which takes additional time for processing. For sure as mentioned about the algorithm previously (Section 2.8) each newly found solution is stored, and when a new solution is found it is compared with the previous one. If the quality was better, then it will be used for comparing the next solutions, while if not, the previously found one will be hold to continue comparing with it to discover the rest possible solutions. This way we can be sure that we will get the optimal solution.

2.11 Summary

Web services discovery and selection is important for solving the web services composition problem inorder to find the best applicable service among similar web services that meet the customer's needs. Web services composition problem is re-using discovered and existing web services and combining them in a process, while web services composition is classified based on three major factors which are the automation degree, operators type (simple or complex), and search set size.

Beam Stack Search algorithm is a heuristic search algorithm that helps in solving the web services composition problem. |It rebuilds each time a new solution is found, which upgrades the solution by discovering better solutions until finding the optimal solution. The cost of response time and the throughput are used to for measuring the complexity of the Enahnced Beam Stack Search algorithm by calculating ratios of time and throughput.

Chapter 3
Related Works

Chapter 3
Related Works

Various research efforts contributed in solving the problem of web services composition depending in different approaches and methods ranging from manual to automatic, syntactic to semantic, non-heuristic to heuristic depending on algorithms such as Beam Stack Search.

In this chapter, we review works related to web services discovery (Section 3.1), web services composition with algorithms that can help in guiding service composition including semi-automatic web services composition approaches (Section 3.2), automatic web services composition approaches (Section 3.3) and using Beam Stack Search algorithm with semantic web services composition (Section 3.4).

3.1 Web Services Discovery

While the web service technology is as well adopted by information technology practitioners and designers, the amount of existing web services is constantly increasing. So, the need of the usual web service discovery method which is based on UDDI record lists, demands more time and persistence by the developer or customer. However, this method is not efficient in many situations because it needs to be able to elect among a great quantity of delivered web services.

Sycara, Klusch, Widoff, and Lu (1999) propose an overview about the dynamic service matchmaking between proxies in exposed information environments. They performed the matchmaking using LARKS in JAVA for proxy advertisements and requests. They implemented the user interface which traces the path of the result set of a request using matchmakers' filters. The filters can be arranged by selecting a checkbox under the desired filters (under control of user). The authors used five different filters in their developed system. The result set pass through the filters from one to another. Upon their study, they concluded that the service matchmaking among heterogeneous software proxies on the internet is frequently done dynamically and must be efficacious.

El Kholy and Elfatatry (2015) present a solution for the web service discovery in service oriented systems using the concept of multi level search. Briefly, we can

describe their system that it receives the customer requirements as an XML file. After that, the requirements pass through three levels of search, where the first level is keyword search which is applied to discover the nominee service. If no matching arised, after that the second level converts the user requirements to formal English language, this phase resolves the problem of unclear sentence building which may be involved in the user requirements. Finally, the third level is that formal sentences are passed to an ontology provider which converts the syntactic words to its domain ontology word. So, the user requirements are reassigned from syntactic to semantic and the second level of semantic search previously takes place. In this search, services are registered with their semantic description.

As Sycara, Klusch, Widoff, and Lu (1999) trace the resulting path using matchmaker's filters, our matchmaking filter is the Beam Stack Search as it discovers the new solution path and filters all the found solutions to decide the best-found solution.

El Kholy and Elfatatry (2015) utilize a keyword search to discover nominee service, while in our research the next nominees are expanded from the current node a level and the WSLA file contains the semantic data ready to use and no need for any conversions from syntactic to semantics.

3.2 Semi-automatic Web Services Composition Approaches

Many previous studies have studied solving web service composition problem using semi-automatic approaches. We present some previous studies as follows.

A semi-automatic approach is presented by Wang, et al. (2011), that includes data mediation and service proposition algorithms to compose web services into an operation by giving service propositions. They define an input/output directed acyclic graph in order to formulate an input/output schema of a web service procedure. Data mediation resolves the heterogeneities between the input and output structures, also transforms a subset of the output structure to the input structure. They developed three data mediation algorithms for output to input matching (leaf-based, structure-based and path-based) to resolve data heterogeneities in the method design. The researchers established a data mediation approach that attempts to automatically discover the best

mappings between outputs and inputs, concluding that path-based data mediation algorithm is the best to use.

Another semi-automatic method (Hu & Wang, 2008) has the advantage of taking the least possible quantity of essential data from the user and saves them in a relational model (relational model uses the basic impression of table as columns and rows), then takes the necessary data to make the composition. At the end of the algorithm a transformation algorithm is applied to map all of the taken data from relational to BPEL model.

Another study (Chan & Lyu, 2008) gives a new semi-automatic approach, it proves the perfection and verifies the correctness of the composed web service by building the model of the web service to be deadlock free. The approach uses WSDL and WSCI (Web Service Choreography Interface) web service files as the base for the method, it takes the information from them to create the web services. The WSDL file defines the login points for each available web service while the WSCI is used for describing the interactions between WSDL operations to accomplish the web service composition using the obtained information.

Another research (Cotfas, Diosteanu, & Smeureanu, 2010) presents a semi-automatic approach where the composition is prepared in a fractal way by using present web service chains that are able to be integrated easily into new web service chains, all web service chains are used as building blocks to create new service chains while they are described using Web Service Business Process Execution Language (WS-BPEL). This way makes it easy generate new and extra complex web service chains.

These methods are semi-automatic while our research applies an automatic algorithm for solving the web services composition problem. Wang, et al. (2011) are using cylicic graph (number of vertices connected in a closed chain) and study the mapping between the web services inputs and outputs to form the composition, while we are not using cylicic graph and we prepare web services associated files having the required data to perform the composition without the need of resolving data heterogeneities. HU and Wang (2008) uses a rational model for data and transform the data from rational to BPEL, while our data are prepared as a graph and Beam Stack

Search heuristic search is used to make the composition. While we used WSDL file as Chan and Lyu (2008) but thy required WSCI file to describe the interactions between web services, while in our study we describe the web services with their functional requirements in the WSDL file only. Cotfas, Diosteanu, and Smeureanu (2010) create blocks of web services chains as BPELs to generate an extra complex web services chains, while we discover a set of separated compositions using the heuristic Beam Stack Search algorithm.

3.3 Heuristic Automatic Web Services Composition Approaches

A lot of previous studies tried to solve semantic web services composition problem automatically such as (McIlraith & Son, 2002), (Sheshagiri, DesJardins, & Finin, 2003), and (Wu, Parsia, Sirin, Hendler, & Nau, 2003). Kil and Nam (2013) proposes using the heuristic Beam Stack Search algorithm (described in full details in Section 2.7) in solving quality aware web services composition problem.

Heuristic, is a function that rates solution path candidates in search algorithms at each branching phase depending on existing data to determine which branch to pursue (Likhachev, Gordon, & Thrun, 2003).

The approaches mentioned in Section 3.2 give a solution where the user has to select the desired service according to quality preferences manually. The reason is that these types of approaches only have the syntactic description of the web services and has no semantic data. Therefore, there is a need to develop the web service composition with the help of the semantic description, which gives the automatic detection for the web services quality for the automatic selection. This led the researchers to start using semantics in their approaches and techniques in different ways.

McIlraith and Son (2002) tackle the problem of automated web service composition and execution for the semantic web. They provided high-level generic actions and modified constraints to address the web service composition problem. As a contribution on an existing ConGolog interpreter, the authors built their implementation and verified the correctness of their work. They use Golog in their study as a natural formalism for solving web service composition problem. Their approach was designed in such a way that it has a possibility to extremely decrease the

search space, also their technique is easy for the usual web user to use and modify. This method has an amplified ability to Golog, which is about allowing to include modifying user constraints. This method contains a programming concept called "order" which gives the ability to relax the notion of sequence and enable the insert process of actions to accomplish the qualification for the next action to perform it by the program inorder to simplify the customization and permitted more generic actions.

Sheshagiri, DesJardins, and Finin (2003) present a planner that composes atomic and basic services which are described using DAML-S into a composite service. While DAML-S is a DAML+OIL that can be employed to supply the semantic description for web services. DAML-S, be made up of a set of ontologies that offer a vocabulary to describe services. A set of services and goal service are given as an input. This planner can dynamically re-plan if a service fails, also the planner is able to produce emergency plans to frustrate such failures.

Wu, Parsia, Sirin, Hendler, and Nau (2003) also uses DAML-S to automate the solution of web services composition problem but composing the planner is in a different way. This system totally plans over sets of DAML-S descriptions using a planner. Consequently, the system accomplishes the subsequent plans over the web. The planner always performs output producing actions as it plans. But this is sometimes is not suitable in some cases such as implementing some web services may take a very long time and the work would be better if the planner continues planning while waiting for this information.

Zhang, Arpinar, and Aleman-Meza (2003) propose a solution for the problem of web services composition. It integrates the use of web services ontologies to help in discovering possible matching between inputs and outputs. Also, this technique is Human-Assisted Automatic Composition system that supplement the Interface-Matching Automatic Service Composition technique through qualifying human participation when the composition cannot continue automatically or when there exist doubts in matching services.

Talantikite, Aissani, and Boudjlida (2009) gives an automatic model for web services discovery and composition problem. This study depends on semantic annotation for web service discovery and composition, inorder to give an

understandable description since it is about assigning names, characteristics, and descriptions. The authors used in their approach an inter connected network representation form for the services set, the semantic web services is represented in OWL-S, and the similarity between two concepts of two services is represented by a connecting edge. Using the similarity measure between the concepts to mark edges, like pellet before any submitted request. To assemble the composition outline of services which satisfies a client's request, the semantic network is discovered in backward chaining and depth-first in a single pass. Finally, a number of composition plans fulfilling the request are obtained, while just one optimal composition plan using quality of services is turned back to the requester. Mainly, this technique decreases the complexity of finding the composition at first, then it decreases the time needed to make the composition design to select the best quality (similarity, time and memory space).

Paikari, Livani, and Moshirpour (2011) presents an automatic frame of work for web service composition P2P network which outlays from an algorithm based on a phased algorithm. This algorithm can match the output of semantic web services of the previous phase with a new one which its inputs must be able to be matched. Multi-agent System Engineering methodology is used to model the frame of work, it is a famous agent directed methodology and a top-down approach. It consists of four agents: UI Provider, Service Finder, Service Provider and Composer. An OWL file has been used to describe web services. The composition procedure is accomplished through a number of steps while the composer directs its request for a suitable next web service to a service finder at each step. The researchers in this study only showed the high-level design and initial implementation of the system and they did not evaluate the performance of the system in a real-world case study.

Other contributors (Qi, Tang, & Chen, 2012) proposed a mechanism to classify web services into diverse categories based on automatic function, then they designed a web service composition system based on service classification and artificial intelligence planning approach which is used for automatic web service composition. The mechanism consists of two main parts: the first part is the service management sub-system which is based on the service classification management mechanism, while the second part is the service provision sub-system, which is used to meet the need of

users' request by artificial intelligence planning. The researchers also concentrated on the classification of web services. In this operation, they compared a single instance of web service with the existing web service categories, by calculating their similarity and comparing their semantic descriptions. After that they developed a design of a service administrative system which is a part of their web service composition system. Finally, referring to the user's request as the input, using artificial intelligence planning engine they created an appropriate composition plan to meet the user's request. This system combines service classification and artificial intelligence planning and workflow, but they did not apply it on the real word or give an implementation results to prove their study by an example.

Another proposed dynamic web service composition algorithm is built based on quality of web services (Yan, Zhijian, & Guiming, 2010), where the use of web services quality component is fundamental since it states the non-functional requirements of service which allow them to work through presenting a hierarchical quality of web services ontology QoSHOnt composed of three layers (upper, middle and lower). While the upper layer outlines the basic impression for defining the specification, context, parameter and relations for the quality of services. The algorithm selects the best service depending on the weight of the quality of web service factory to get the best web service. For arbitrary web service request, the researchers worked on getting the best immediate descendant web service or service set by invoking a function that works on getting the maximum quality of service in the service composition map. In this study, there was an absence for relevant standard platform and standard test data sets, therefore the researchers used a random replicating web service as a test case, they also chose six data sets (300, 600,900,1200,1500 and 1800 service) with 30 random requests for each data set to make the composition of web service. Also, they used average time of the combination to calculate the experiment results of web service composite efficiency.

A study presented by Yan, Xue, and Yao (2009) that explores web services ontology and Ant Colony algorithm. The researchers here, proposed a method of web services composition that is based on Ant Colony algorithm which helps to ensure to get the best composition of web services in a less time. This project has two benefits, the first one that it has a high successful rate of services composition, while the second

benefit that it ensures the quality of composition and the efficiency for the composition of web services which is based on the users' requests in the field of dynamic composition of web services. The researchers used OWL-S for the description of web services and their relationships. They converted the composition of web services into a classic graph theory problem, and solved the composition problem by using the benefits of Ant Colony algorithm. They showed that the algorithm is fruitful to compose guaranteed quality and efficient web services. They concluded that they need to enhance the Ant Colony algorithm, parallel composition of web services and the services quality of services control problems so that the algorithm may be well adjusted to a parallel composition of web services.

Another study also used the Ant Colony algorithm for solving web services composition problem is due to Srour, Othman, and Hamdan (2013). This study presented a user amiable and efficient automatic web services composition model using Ant Colony System. Their model depends on four core components (Visual Services modeling, User Query Generator, Semantic Composer and Workflow Generator). The model works on automating the composition procedure with taking into account the end user viewpoint seeking to decrease the exploration space of candidate Web services, and it also improves Web Services composition usability and efficiency. The researchers applied backward discovery strategy for web services selection and Ant Colony System for web services composition process to make the web services composition automatically, but they did not make any evaluation of the model.

3.4 Beam Stack Search with Semantic Web Services Composition

Some studies (Marshall, 2016) referred to heuristic search methods as methods which might not always find the best solution, but these methods try to find a good solution under a practical time by deciding which choice might be the best one. Kil and Nam (2013), as disscussed next, adopted this idea and used specific time thresholds to study efficiency.

Kil and Nam (2013) propose a solution for the web service composition problem using the Beam Stack Search algorithm. This study gives a dynamic search methodology to solve the web services composition problem using the time quality

factor. They implemented the algorithm using some computations to give different beam widths through the search process that change as a trio form which is different from Beam Stack Search which uses a fixed beam width. Kil and Nam used C++ language to program the algorithm, also they used specific time thresholds in their experiment to study the efficiency of their work using four different thresholds in seconds.

In our study we use the Beam Stack Search algorithm with different fixed beam width sizes to solve the problem of semantic web services composition. We suppose that the heuristic search algorithm passes through all of the levels of search tree and is able to discover all of the possible paths of solutions and terminates itself regardless of the additional time. We try to find the best solution among all the available candidate solutions inorder to measure the efficiency through taking the quality ratio between the first found solution and the optimal solution.

In our work, the quality factor will be wider than the previous search algorithm investigated by Kil and Nam (2013). The quality factor depends on time and throughput so that it can be used to find two quality ratios which are the ratio of "the first fast solution throughput" to "the best throughput solution" as well as the ratio of "the first fast solution time" to "the best throughput solution's time" are used to assess the Enahnced Beam Stack Search algorithm efficiency which help us to study the efficient of waiting more time inorder to get better throughput value for web service composition problem solution.

Kil and Nam (2013) study web services on large scale sets, they use six different sets of web services with sizes 50, 100, 100, 500, 1000, and 1500. In our case, we consider a larger sets of web services. We apply the Enahnced Beam Stack Search algorithm on sets of 5000, 10000, and 15000 web services with four different beam widths in our experiments (120, 150, 300, and 600).

3.5 Summary

The usual web service discovery methods such as (Sycara, Klusch, Widoff, & Lu, 1999) and (El Kholy & Elfatatry, 2015), request from the customer or the developer a supplementary time and persistence. Therefore, we cannot depend on these

methods all the time since they request to be able to choose among excessive delivered web services.

Semi-automatic approaches such as (Wang, et al., 2011), (Hu & Wang, 2008), (Chan & Lyu, 2008), and (Cotfas, Diosteanu, & Smeureanu, 2010) do not adapt to choose the best web service under the user constraints without a human involvement in the part of web services semantics. Automatic approaches such as (McIlraith & Son, 2002), (Sheshagiri, DesJardins, & Finin, 2003), (Wu, Parsia, Sirin, Hendler, & Nau, 2003), and (Kil & Nam, 2013) use web services semantics to find a solution under the specific time constraints of the algorithm.

In the next chapter, we present our enhancement on Beam Stack Search algorithm with diverse fixed beam width sizes to solve the problem of semantic web services composition and record the spent time with the discovered optimal solution.

Chapter 4
Composing Web Services Semantically Using Enhanced Beam Stack Search

Chapter 4
Composing Web Services Semantically Using Enhanced Beam Stack Search

In this chapter, we present the approach for composing web services based on their syntactic as well as semantic descriptions using Beam Stack Search. The composition is formed based on user's goal, i.e., a less optimal fast solution, or an optimal slower solution.

Section 4.1 presents a specification of the web service composition. Section 4.2 briefly covers the structure of the web services composition approach using Enhanced Beam Stack Search algorithm. Section 4.3 describes the Enhanced Beam Stack Search algorithm, while the factors of web services quality are defined in Section 4.4. Section 4.5 shows the base of measuring qualities of the experimental results. Finally, Section 4.6 describes the case study used in the experiments.

4.1 Specification of Web Services Composition

Web services composition consists of n number of required services tasks expressed as ($Service_1$, $Service_2$, … , $Service_n$) as shown in Figure 4.1 where $Service_1$ is considered as the start node and $Service_n$ is the goal node which we have to set them in our Enahnced Beam Stack Search algorithm application.

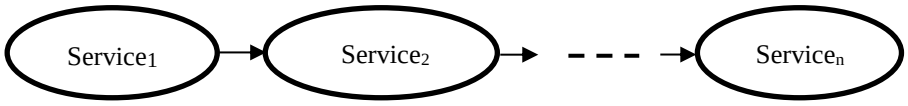


Figure (4. 1): Web services composition

As discussed in Section 2.5, composition can be based on large scale set of web services using simple operators. The set web services is represented as a graph to form the web service composition using such simple operators as shown in Figure 4.2:

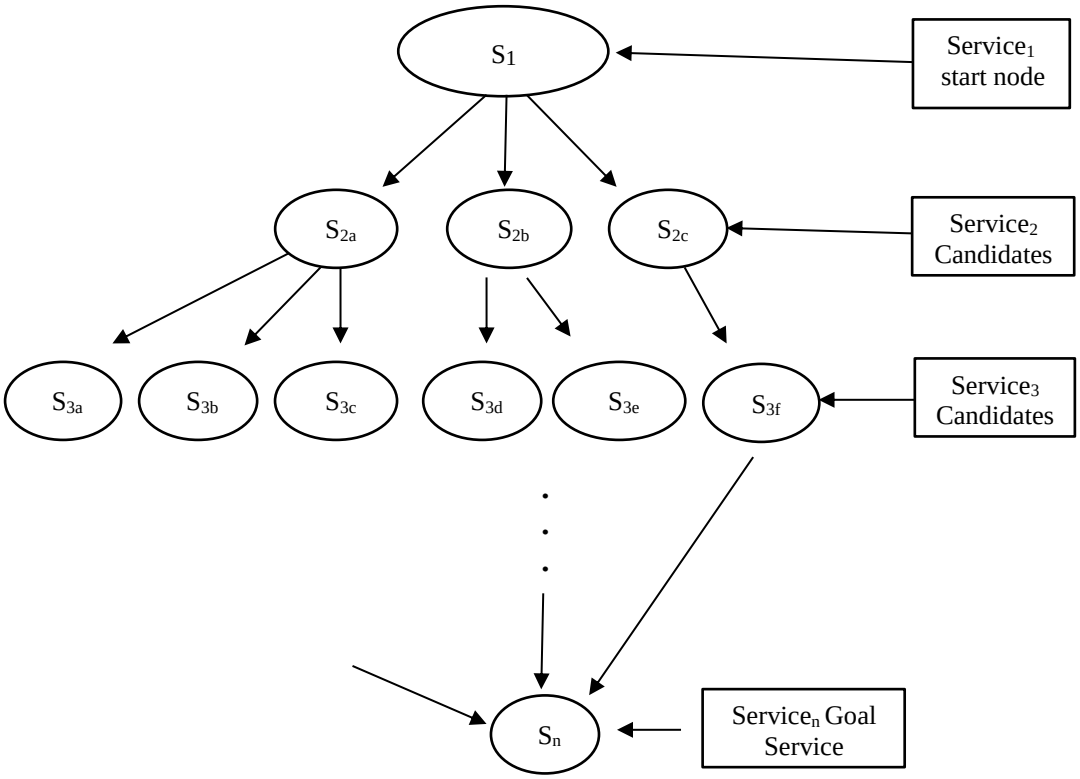


Figure (4. 2): Directed graph composed of candidate services

There are n candidate automatic services with the same functions but different qualities for each service task which are generally known by the non-functional requirements. The problem of web services composition can be resolved by a graph with different candidate services paths. Finding the optimal web services composition is the problem of finding the optimal path in the graph. The composite web service that is composed of the optimal path, should be the best path meeting the requirements of the user. User requirements are usually the non-functional requirements such as response time, security, throughput. In our research, we focused on the response time and throughput to study the results of our experiments depending on them.

The goal node (S_n) can be reached through multiple paths using Beam Stack Search algorithm. We have to notice that by increasing the beam width, the algorithm may prune some node (service) at early level while this node has the possibility to give a path with higher quality value. For example, if the beam width for the graph in Figure 4.2 is 3, and S_{2c} throughput is 5 but the highest path throughput under this node is 10, but S_{2b} throughput is 3 which will be expanded after S_{2c} and it may have a path to S_{3f}

with a higher quality which will not be taken into account since S_{3f} was previously discovered through S_{2c}. In the other hand if the used beam width is 2 then it will take the path discovered by S_{2b} because it is discovered before the path from S_{2c}. This indicates that a smaller beam width leads to a better solution.

The quality measure, either throughput or response time, is calculated for each transition from one service to another until discovering all of the possible web services compositions. The total quality measure of the composition is calculated by:

$$\sum_{i=1}^{n} \boldsymbol{Transition}(\boldsymbol{S_i}, \boldsymbol{S_{i+1}}) \qquad (4.1)$$

Equation 4.1 calculates the summation of all the transitions between web services nodes in each web services composition flowing from the first service S_1 to the goal service S_n.

Based on the specification of web services composition, we present our approach to accomplish this composition using a modified version of Beam Stack Search.

4.2 Structure of Web Services Composition Using Enhanced Beam Stack Search

In this section, we present our proposed web services composition solution using Enhanced Beam Stack Search. We start by giving an overall description of how the approach works. Then we explain in detail each part of the approach.

Figure 4.3 illustrates the proposed approach for web services composition. In the set of web services, each service has two descriptions; syntactic description (in WSDL) and a semantic description (in WSLA), specifically the qualities. The throughput and the response time are used as the quality measures. The throughput is recorded from the semantic description (WSLA) and measured by Equation 4.1. The transition in the equation is considered as the quality factor. Thus Equation 4.1 is used as $\sum_{i=1}^{n}$ Throughput $(\boldsymbol{S_i}, \boldsymbol{S_{i+1}})$ which sums the throughput values for all nodes used in the composition. The response time is the required time spent to move from the start web service node to the next one until reaching the goal web service node, which is calculated based on Equation 4.1 as $\sum_{i=1}^{n}$ Time $(\boldsymbol{S_i}, \boldsymbol{S_{i+1}})$.

The Beam Stack Search algorithm is a search graph which means that it must be able to search as a graph of web services. Each node of the graph has an ID which represents a web service with its corresponding semantics.

Our enhancement on the Beam Stack Search is about extending it to able to deal with web services based on their syntactic descriptons in terms of WSDL and in terms of their semantic description in terms of WSLA organized in what is called web services pool. In the service pool, each web service is represented with a special ID (based on a hashmap structure).

The Enahnced Beam Stack Search algorithm creates the solution search space as a graph of web services like the graph shown in Figure 4.2 which is processed by the Enahnced Beam Stack Search algorithm in order to start solving the problem of web services composition and find a set of candidate solutions which meets the user request.

Two solutions in the set of solutions are the most special for a user, the first one is the first found solution which a user choose it when he is interested in the time quality and do not want to waste time, while the other solution is the optimal solution that has the best-found throughput quality among all of the possible found candidate solutions in the set of solutions which is preferred for users who have flexibility in time and they care about throughput quality.

Next, we elaborate each part of the Enhanced Beam Stack Search algorithm structure to fully discuss it:

4.2.1 Defining the set of web services with their semantic descriptions

The dashed part in Figure 4.3 represents this part of the approach. Firstly, we use a set of web services with syntactic descriptions (in WSDL format) together with their associated quality information (in the WSLA format). This quality information includes the throughput of the service. The set of these services form a pool of web services arranged such that each web service has an ID refering to its full available data. This way we prepare these web services in a way to be ready to form a graph to be searched by the Beam Stack Search as a solution search space.

4.2.2 Forming the web services Graph

The web services still need some arrangement to be ready for use in the Beam Stack Search algorithm since the algorithm can not decide which web service comes before or after the other one. We perform this step of forming web services tree graph as an important phase to be added to the algorithm. The Beam Stack Search algorithm is working with simple operators as a tree graph. We develop a tree graph of web services using IDs from the service pool to denote each node. The graph nodes have input instances, output instances and throughput quality for each service while the response time is calculated through processing the search procedure. We discussed such a graph in the specification of the composition of web services in Section 4.1

4.2.3 Composition problem

Web services composition problem is about composing a sequence of services that give a final service for the client under his specified requirements as specified in Section 4.1 (Specification of Web Services Composition). The client request contains the required web service descriptions (as WSDL). Figure 4.3 shows that there is a relationship between the set of web services Service(input instances, output instances, quality) and the composition problem, this relation comes from that the required services is originally a subset of the set of web services. This set of web services and the formed graph are given as parameter to the Beam Stack Search algorithm to match instances from the two sets in finding the solution.

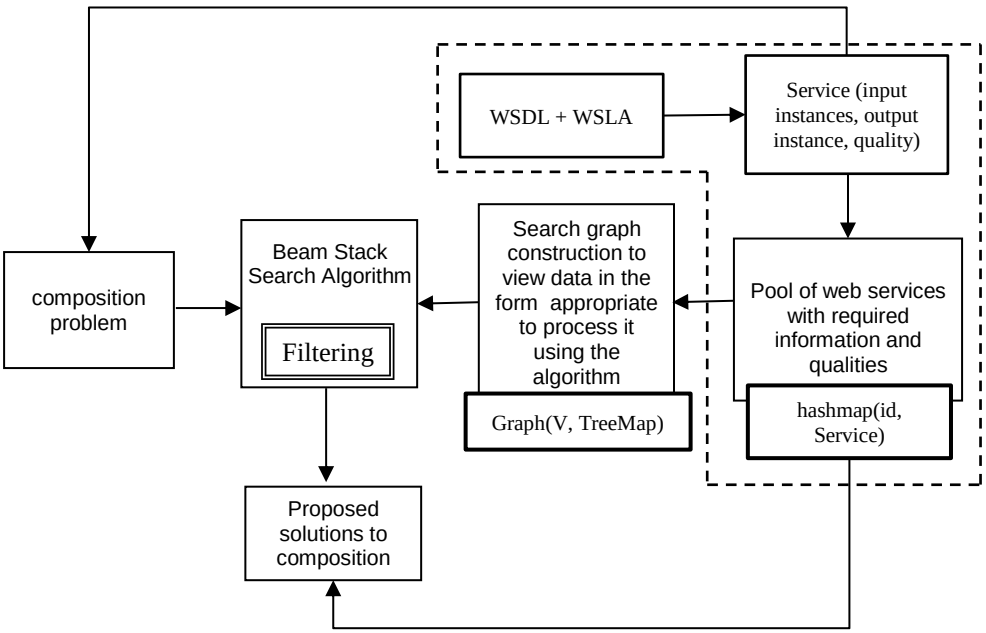
4.2.4 Beam Stack Search

Our study aims to solve the web services composition problem by discovering and selecting web services automatically. As from the previous steps now we have an automatically formed search graph of web services and a search problem which needs to be solved. Then, we are now prepared to use the heuristic Beam Stack Search algorithm to solve the problem of web services composition. We discuss these details in a separate section due to its importance (see Section 4.3).

4.2.5 Proposed solutions

The Enahnced Beam Stack Search algorithm discovers a set of candidate composite solutions which all have the same functional requirements but differ in their non-functional requirements, specifically the throughput, all of the found solutions are correct and the user can take any one of them, but only two solutions are special. The first one is the first found solution since it is considered as the fastest solution. Also, the Enahnced Beam Stack Search algorithm filters all of the found solutions and specify the optimal solution which is the solution with the best non-functional requirements (best quality) and this is considered as the second special solution among the rest of found solutions because it overcomes all of the rest solutions with its non-functional requirements.

The user can choose the first solution if he is concerned with time by terminating the Enahnced Beam Stack Search algorithm after giving the first solution. If the user is more interested in the non-functional requirements and he has the ability to wait for more time, he will choose to wait for the algorithm to continue processing and filtering until finding the optimal solution. The Enahnced algorithm stops by printing the candidate solutions with their calculated qualities, throughput and response time. Figure 4.3 gives an illustration of the approach structure.



Abbreviations: V = number of vertixes of the graph, TreeMap= inputs of the graph using data from the service pool

Figure (4. 3): Web service design "Enhanced Beam Stack Search"

Next, we present the Enhanced Beam Stack Search algorithm (as a continuation to Section (4.2.4) together with its usage in finding the solution of the composition problem.

4.3 Enhanced Beam Stack Search Algorithm

For the Beam Stack Search algorithm to be suitable to solve the web services composition problem, it has to be enhanced to deal with the composition problem as well as the web services syntactic as well as semantic descriptions in terms of WSDL and WSLA. Therefore, the Enahnced Beam Stack Search algorithm considers the following issues:

1. **Web services pool:** creating a pool of web services include, their non-functional requirements, i.e., their syntactic as well as semantic descriptions in terms of WSDL and WSLA

2. **Input graph:** represents web services in the web services pool and creates a search space which is formed as a search graph for the Beam Stack Search algorithm.
3. **Solving the composition as a search problem:** searching through the graph of web services to form a composition using Beam Stack Search.
4. **Solutions:** extract a set of possible solutions based on the user request and filter the set of solutions to find the optimal solution (best throughput value). The user can choose the first found solution if he prefers to preserve time.

The Enhanced Beam Stack Search algorithm automatically processes the web services while solving the composition problem. Function DGS, shown in Figure 4.4, is used by the search function, showin in Figure 4.6, to build a graph search tree of web services. The search function, then, is used by the Beam Stack Search algorithm (Figure 4.5) to find the required composition solution.

The inputs of "Define a graph search tree of web services" function are the web services data in WSDL and its associated WSLA (line 2 in Figure 4.4). The function process these syntactic and semantic description of the web services (line 4 in Figure 4.4).

After obtaining the required data, the function stores them in a service pool (line 5) with each web service having an ID referring to it. By executing this in **Function DGS**, we achieve the dashed part in Figure 4.3 (which illustrates the design of the Enhanced Beam Stack Search) additionally with the part of the next step of the search graph construction. To form the web services graph in line 10 at Figure 4.4 (also see Figure 4.2) it depends on matching each web service output instance with the suitable input instances among the available web services in the servies set.

Now, in order to execute the search graph part, the function prepares for this step by setting the graph nodes at first as the service ID (line 7 in **Function DGS**). After that, the function maps from each node to the next possible nodes (line 8 in **Function DGS**) to form the search graph (line 9 in **Function DGS**) which will be used by the function Search, in the Beam Stack Search (see Figure 4.6).

Next, Figure 4.4 includes the pseudocode for the dashed part of Figue 4.3

```
1   Function DGS: Define a graph search tree of web services
2   Input WSDL and WSLA files
3   Output pool of web service generated as a graph search tree
4     Translate WSDL and WSLA to Service(input instance, output instances, quality)
5     service-pool = hashmap(id,Service)
6   While Services is not null
7     then
8       vertex v;
9       v = service(id)
10    Map = forms a graph from each v to the next neighbor vertices // as edges from v to v
11  Else
12  Return Graph(V, Map)
13  End
```

Figure (4. 4): Define a graph search tree of web service function

The output of this function in Figure 4.4 is used in the inputs of the search function specified in Figure 4.6.

The Beam Stack Search algorithm is executed on the resulting graph as shown in Figure 4.5.

```
1   Algorithm BSS: Beam Stack Search (set webServices, Service requiredService,
2   Real U)
3   Output: The first found solution and the optimal solution with their non-functional
4   requirements
5   Beam_stack = Ø;
6   Beam_stack.push([0,U)); // initialize beam stack
7   bestPath = null; // initialize optimal solution path
8         while Beam_stack.top() ≠ null;
9             solution-path = Search(webServices, start, goal, U, beamWidth);
10            if goal-path ≠ null then  //  solution path
11                best-path =  goal-path;
12                U = Cost(goal-path);
13                return goal-path;
14        While Beam_stack.top().fmax > || = U do
15        Beam_stack.pop();
16        End while
17        If Beam_stack.isEmpty() then
18        return bestPath is the optimal solution
19        Beam_stack.top().fmin= Beam_stack.top().fmax;
20        Beam_stack.top().fmax=U;
21  End while
```

Figure (4. 5): Beam Stack Search Algorithm

The inputs (line 1 in **Algorithm BSS**) are the set of web services, required service, and the upper bound U (the upper limit for the quality, i.e., response time). Also, at line 6 in **Algorithm BSS** (where it uses the search function), we need to look at the inputs of Function Search where it uses the graph prepared in Function DGS as an input, besides the start node of the search, the goal node, the upper bound U, and the beam width value. The goal node is the required web service by the user. The cost function at line 12 in Figure 4.5 represents the total quality measure which is calcuated using Equation 4.1 which calculates the summation of all the transitions between web services nodes in each web services composition from the first service to the goal service.

The Beam width plays a big role in the search results, it affects the quality values (response time and throughput). A smaller beam width size makes the response time of the first found solution shorter. For an optimal solution, beam width size impacts the throughput quality value such as increasing the beam width decreases the throughput while decreases the response time. In this case, the user who cares about throughput quality ignores this time decreasing.

The beam width value is important in the search process and choosing it is critical depeding on the user. A user who cares about time and intends to choose the first found solution prefers to use smaller beam width size to get a faster composition. While a user who cares about throughput and intends to choose the optimal found solution, the smaller beam width is also the best choice. This is proved by the experiments preseted in Chapter 5.

```
1    Function Search: Search (Graph webServices, Node start, Node goal, Real U,
2    Integer beamWidth)
3    Best-goal = null;
4    open[0] = {start};
5    l =0; // start level while l is the index of layer
6    open[l] = Ø;// index of level
7    closed[0] = Ø;
8    w = beamWidth;
9          while open[l] ≠ Ø; do
10           node = argmin_n{ cost(n) | n ϵ open[l] }
11           open[l] = open[l]\{node}
12           closed[l] = closed[l] U {node}
13                 if (node = goal)
```

```
14                  if cost(node-path) < cost(best_ composition)
15                  then best-goal = node;
16                  set U = g(best-goal); // g(node) is the cost of the best-goal path from
17                  the start node to the goal node
18                  End;
19              Else
20              Node.expand(beam-stack.top()) // top level workflow automatically
21              If layerSize(l+1)>w then
22                  Keep = keep the best w nodes ϵ open[l+1];
23                  Prune = {n | n ϵ open[l+1] && n ∉ Keep };
24                  Beam_stack.top().f_max = min(cost(n) | n ϵ prune };
25                  For each n ϵ Prune do
26                      open[l+1] = open[l+1] \ n
27                      delete n
28                  then Keep = open[l+1]; // after the pruning
29              End
30      End while;
31      if 1 < l≤ relay or l> relay + 1 then
32        for each n∈ Closed[l−1] do /* delete previous layer */
33          Closed[l−1]←Closed[l−1]\{n}
34         delete n
35        end for
36      l= l+1; // move to the next level
37      Open[l+1] = Ø
38      Closed[l] = Ø
39       Beam_stack.push([0,U)); // new item in the stack
40      till
41          If best-goal ≠ null; then
42          Return best-goal-path;
43          Else
44          Return null;
45
    End if
```

Figure (4. 6): Search function

By these three parts together, we developed an Enhanced Beam Stack Search algorithm to solve the web services composition problem according to the user request.

4.4 Factors of Web Services Quality

Through studying the different available web services composition candidates for solving the problem, we focus on two major factors which are response time and throughput. They are defined as follows:

4.4.1. Throughput

Throughput is the quantity of efficiency produced over time through a test. Also, it's expressed as the degree of clarity that can be handled by a web service. Throughput values are gathered from the related WSLA file for the web services non-functional requirements. The user may specify a throughput goal that he needs before starting a test and search for it or he may search for the best possible found throughput as we do in our research.

As an example, on the throughput from WSLA file at Figure 4.7 where it shows that the throughput value for the web service is "20" throughput.

```
<Metric name="ThroughputMetric" type="long" unit=" Throughput">
<MeasurementDirective name="Throughput" resultType="long"
xsi:type="GenericQoSDimension">
<Value>20</Value>
</MeasurementDirective>
</Metric>
```

Figure (4. 7): Throughput example

4.4.2. Response Time

Response time is the elapsed time between the start of the search and finding the required web service. In our study, the Enahnced Beam Stack Search algorithm calculates automatically the response time of finding each solution separately to use it in the evaluation for each found solution.

As an example, on the response time from WSLA file at Figure 4.8 where it shows that the unit of response time value for the web service is "5" milliseconds.

```
<Metric name="ResponseTimeMetric" type="long" unit=" milliseconds">
<MeasurementDirective name="ResponseTime" resultType="long"
xsi:type="GenericQoSDimension">
<Value>5</Value>
</MeasurementDirective>
</Metric>
```

Figure (4. 8): Response time example

Next, we describe the case to be used in the experiments and evaluation covered later (in Chapter 5). There we perform a number of experiments and use the results in evaluating the quality of the throughput and the quality of response time.

4.5 Measuring Qualities

The cost of response time and throughput are used to measure the qualities of a given web services composition solution. This is done by taking the ratio of time as well as throughput between the first found solution and the optimal solution as shown in Equations 2.1 and 2.2. This shows how the proposed solution improves finding the optimal solution. The Enahnced Beam Stack Search algorithm finds a fast solution followed by a number of candidate solutions until finding the best solution, the user can take the first solution or wait for the best solution to be found. The other candidate solutions can be used as sub-optimal solutions which are for sure better than the first found solution but not the optimal.

$$\boldsymbol{Quality\ Ratio_{Throughput}} = \frac{\textbf{Optimal solution throughput}}{\textbf{First Found solution throughput}} \qquad (2.1)$$

$$\boldsymbol{Quality\ Ratio_{Time}} = \frac{\textbf{Optimal solution response time}}{\textbf{First Found solution response time}} \qquad (2.2)$$

4.6 Case Description

We used sets of web services with their functional and non-functional properties collected and prepared by Blake, Weise, and Bleul (2010) for the Acme Packet company services to represent our case. Acme Packet provides control functions to

deliver trusted, interactive communications voice, video and multimedia sessions across IP network borders (Wikimedia Foundation, Acme packet, 2016).

Each set of web services has its functional requirements in terms of WSDL and their associated non-functional requirements in terms of WSLA. We use three different test set sizes in our experiments: 5000,10000, and 15000. Each test set has a user request in terms of WSDLs which forms the composition problem. We use four different beam width sizes: 120, 150, 300, and 600 inorder to study the impact of small and big beam width sizes on the quality of web services (the non-functional requirements).

4.7 Summary

Web services composition consists of n number of required services tasks expressed as ($Service_1$, $Service_2$, … , $Service_n$) where $Service_1$ refers to the start node and $Service_n$ refers to the goal node.

The enhancement on the Beam Stack Search is made to allow the algorithm to deal with web services based on their syntactic (WSDL) and semantic (WSLA) representations stored in web services pool. The Enahnced Beam Stack Search algorithm creates the solution search space as a graph of web services that is processed by the algorithm in order to solve the problem of web services composition by finding a set of candidate solutions which meets the user request.

Two important solutions in the set of solutions are special for the user:

- The first found solution, users choose it when they are interested in response time quality
- The optimal solution, that has the best-found throughput quality among all of the possible found candidates of solutions in the set of solutions. This solution is preferred for users who have flexibility in time and they care about throughput quality.

Chapter 5
Experimental Results and Evaluation

Chapter 5
Experimental Results and Evaluation

While the Enahnced Beam Stack Search algorithm gives a set of solutions, the first found solution is defined as the main focus besides the optimal solution. The user can choose between both of them, since he can take the first found solution with less response time when he does not have time to wait for the Enahnced Beam Stack Search algorithm to continue the computations until finding the optimal solution. This is considered a good solution but the throughput quality value predefined in Section 4.4 is not be the best. In case the user has enough time to wait for the Enahnced Beam Stack Search algorithm to finish computations, the optimal solution could be his choice.

In this chapter, we talk about some implementation issues in Section 5.1. While in Section 5.2 we view our experimental results, and give an evaluation for these results in Section 5.3

5.1. Some Implementation Issues

Beam Stack Search algorithm is employed in this study to solve the problem of web services composition automatically. The algorithm searches through a large set of web services and finds a set of candidate solutions using a stack that helps in reducing the used memory space. It avoids nodes re-expansion in each loop by storing the unused values of the expanded nodes in the closed list in order to use them the next loop. Reducing the re-expansions help in reducing the required time in computations.

Web services are designed using WSDL orchestration, while there are two used WSDL files; the first one contains the set of web services, and the other one contains the client's web service request which is a subset of the first full file. The WSDL file contains the syntactic description of the services and has an associated WSLA file for the services semantic data.

These WSDL and WSLA files were generated previously by Blake, Weise, and Bleul (2010). These WSDL and WSLA files are interpreted to Java using JDOM parser in order to use them in our implementation. The data is processed after that using Service Java file class to describe each service and the associated data, then all of the

web services data are stored from the Service in a ServicePool generated to hold the web services data associating each web service with a special ID which is used as nodes in the search graph after that.

We use a WSDLParser.java class which depends on using the SAXBuilder (Hunter & McLaughlin, Class SAXBuilder, 2015) to read the data from the WSDL file, the SAXBuilder is implemented by the jdom.jar (Hunter & McLaughlin, JDOM, 2000) build library which we use in our implementation. WSDLParser.java class mainly loads the data from the set of services WSDL file to a file input stream with the help of the SAXBuilder then we used another class calling it as the service pool class to load the data in a map to arrange the web services with their IDs.

Each web service is connected with multiple web services in the next level of the search graph, all having similar functional requirements but differ in the non-functional requirements. The algorithm uses them as candidates to find the solution.

The WSDL composition problem file which contains the required web service requirements is also read to Java using the Service Java file because this Java class describes it well as a subset of the set of web services, then the required service is used by the Beam Stack Search algorithm as the input to be search for.

5.2. Experimental results

Several experiments are performed to the Enahnced Beam Stack Search algorithm using 10000 web services set and 4 different beam widths; 120, 150, 300, and 600 respectively. We set the upper cost limit in our experiment to null since we are searching for the highest throughput value. This gives a throughput qualities for the first found solution 9, 9, 5, and 7 respectively with response times 1212, 1900, 2199, and 2889 milliseconds correspondingly. This can be an accepted solution when the user is interested in a short response time. Having a look at the response time quality, in the case of using the first found solution, time quality value is high while the throughput quality is low. For the user who is interested in throughput quality, the Enahnced Beam Stack Search algorithm continues processing until concluding with the optimal solution. In our experiments using the same consequent beam widths 120, 150, 300, and 600, the obtained throughput values are 796, 639, 301, and 180

respectively as optimal solutions' throughput values for each used beam width size correspondingly. The values of the throughput quality are decreasing when increasing the beam width value as shown in the results. While the response times are 307573, 301167, 312150, and 361757 respectively for each beam width which results in consuming more time to get the solution of the problem because it needs to do more computations which require additional time. Although the user who is concerned with throughput quality is also interested in the response time, but he has to give some compromise in this case by accepting the additionally spent time in calculations in order to get the best throughput quality.

Quality ratios give better understanding for the results based on the following equations:

$$\boldsymbol{Quality\ Ratio_{Throughput}} = \frac{\textbf{Optimal solution throughput}}{\textbf{First Found solution throughput}} \quad (5.1)$$

$$\boldsymbol{Quality\ Ratio_{Time}} = \frac{\textbf{Optimal solution response time}}{\textbf{First Found solution response time}} \quad (5.2)$$

A higher difference between the first found solution and the optimal solution means that we have a high throughput quality ratio. The larger is the throughput quality ratio, the better experimental results. This is because searching for the optimal solution gives a better reward than the first found solution which means that it worth searching for. So, whenever we have an optimal solution throughput quality higher than the first found solution throughput quality.

On the other hand, having a high time quality ratio is not preferred, but it does not affect the results as we get a better throughput quality ratio through the increased time. In short, when the user prefers high throughput he naturally spends more processing time.

Table (5. 1): Qualities and quality ratios for 10000 web services test set

		Results and calculations					
Beam width	**test set size**	**ffsTH**	**osTH**	**ffsT**	**osT**	**THqr**	**Tqr**
120	10000	9	796	1212	307573	88.44	253.77
150	10000	9	639	1900	301167	71	158.50
300	10000	5	301	2199	312150	60.2	141.95
600	10000	7	180	2889	361757	25	125.218

Table 5.1 includes the quality ratios calculated for the experiments using the different beam widths. We use some abbreviations in the table such as: ffsTH denotes to first found solution throughput, osTH denotes to optimal solution throughput, ffsT denotes to first found solution response time, osT denotes to optimal solution response time, THqr denotes to throughput quality ratio, and Tqr denotes to response time quality ratio. Time is calculated in milliseconds.

Here is an example of calculating quality ratios for a test set size of 10000 with a beam width of 120. The first found solution throughput is 9 with first found solution response times of 1212 milliseconds and the optimal solution throughput is 796 with optimal solution response times of 307573 milliseconds.

$$\boldsymbol{Quality\ Ratio_{Throughput}} = \frac{\mathbf{796}}{\mathbf{9}} = 88.44$$

$$\boldsymbol{Quality\ Ratio_{Time}} = \frac{\mathbf{307573}}{\mathbf{1212}} = 253.77$$

Calculating the quality ratios for each used beam width of 120, 150, 300, and 600 with the 10000 test set size, give throughput quality ratios of 88.4, 71, 60.2, and 25 respectively which show that increasing the beam width decreases the throughput quality ratio. The response time quality ratios are 253.77, 158.50, 141.95, and 125.218 which is decreasing respectively by increasing the beam width, when response time quality ratio is high this means that we wait for a long additional time after finding the first found solution. Since the optimal found solution throughput values deserves waiting for, then it is not considered a weakness in our results.

The Beam Stack Search is designed to solve big size problems, we use different test set sizes to obtain all kinds of results and checks the efficiency in solving the web services composition problem. This is performed by repeating the experiment with the same variable values of the beam width sizes and with different test set sizes. We get, analyze and compare the results of using smaller or bigger web services test sets in giving better or worst throughput values and response time by the Enahnced Beam Stack Search algorithm.

By performing the experiments on the 5000 and 15000 test set sizes, each result could be analyzed separately for the sequential test set sizes 5000, 10000, 15000.

Using same beam widths of 120, 150, 300, and 600. The first found solution throughputs using the 5000 test set size are 6, 5, 4, and 4 respectively. The optimal solution throughput values are 723, 470, 285, and 161 correspondingly. On the other hand, using the 15000 test set size gives a first found solution throughput values as 10, 10, 9, and 10 while the optimal solution throughput values are 848, 773, 311, and 192 with the same used consequent beam widths.

Table (5. 2): Qualities and quality ratios for 5000 web services test set

		Results and calculations					
Beam width	**test set size**	**ffsTH**	**osTH**	**ffsT**	**osT**	**THqr**	**Tqr**
120	5000	6	723	1745	198795	241	251.06
150	5000	5	470	2809	212376	117.5	156.57
300	5000	4	285	3236	214878	57	136.24
600	5000	4	161	3650	287068	40.25	123.67

Table (5. 3): Qualities and quality ratios for 15000 web services test set

		Results and calculations					
Beam width	**test set size**	**ffsTH**	**osTH**	**ffsT**	**osT**	**THqr**	**Tqr**
120	15000	10	848	780	438105	84.8	254.87
150	15000	10	773	1091	439812	77.3	194.66
300	15000	9	311	1272	440877	34.55	168.93
600	15000	10	192	2138	451384	19.2	134.27

The full detailed throughput values are displayed in Table 5.2 and Table 5.3 related to the corresponding response time values at each beam width for 5000 and 1500 test set size respectively with their throughput quality ratios and time quality ratios.

In all the cases of test set sizes the first found solution response time is trivial which do not make problems even when changed. For the optimal solution, the real waiting time keep increasing by the size of the set test, this is because as an example the 15000 test set size requires more processing rounds which require more time, but this additional time is spent to give better solution throughput values meeting the user requirements of better throughput.

The differences between the throughput values of the first found solution cannot be noticed for the different used beam widths, even sometimes they have similar values with a little difference in the response time.

5.3. Evaluation

Comparing the results related to the test set size using two close beam width values such as 120 and 150. In the case of using 10000 test set size, as shown in Figure 5.1 that the first found solution throughputs were the same quality equaling 9 with response times 1212, and 1900 milliseconds for each beam width respectively. In the case of using 5000 test set size, as shown in Figure 5.2 that the first found solution throughputs equals 6 and 5 with response times of 1745 and 2809 milliseconds for each beam width respectively. In the case of using 15000 test set size, as shown in Figure 5.3 that the first found solution throughputs were the same quality equaling 10 with response times of 780 and 1091 milliseconds for each beam width respectively. For the first found solution, two different closed beam width values give a nearby or even the same first found solution throughput values, and the value for the smaller beam width size is obtained faster. For the optimal solution throughput values, increasing the beam width with small amount affects throughput badly by decreasing it, while the response time decreases when increasing the beam width. The user who cares about throughput quality value will ignore this decrease in time. Choosing the beam width plays a big role in the search process for each user. A user who cares about time and intends to choose the first found solution will prefer to use less beam width to get it faster. While who cares about throughput value and intends to choose the optimal found solution, the less beam width will be the best choice to get the higher throughput value.

Studying two numbers as 300 and a double number as 600 for beam width values, the first found solution throughput values using the 10000 set size as shown in Figure 5.1, gives 5 and 7 consequently with response times 2199 and 2889 respectively. Using a 5000 test set size as shown in Figure 5.2, gives the same throughput value equaling 4 while the response time at beam width 300 is 285 milliseconds and decreased to 161 milliseconds using 600 beam width size. Also, the 15000 test set size results as shown in Figure 5.3, were compatible with our previous

results since they gave a throughput of 9 at 311 milliseconds with 300 beam width size, and 10 first found throughput value at 192 milliseconds with 600 beam width size. The results indicate that a beam width of 600 with higher throughput value, is not a big difference to force the user to choose it.

Viewing the optimal solution results using 300 and 600 beam widths, for the 10000 test set size the throughput values as shown in Figure 5.1 are 301 and 180 with response times 312150 and 361757 respectively. For the 5000 test set size as shown in Figure 5.2, the throughput values are 258 and 161 with response times 214878 and 287068 respectively. For the 15000 test set size as shown in Figure 5.3, the throughput values are 311 and 192 with response times 440877 and 451384 respectively.

The results show that doubling the beam width give about half of the throughput value with a higher time which is not preferred while it is decreasing the throughput also is not favored for the response time to be increased, then choosing a smaller beam width is also better in this case.

Generally, decreasing the beam width takes more time in computations since the number of loops in the Enahnced Beam Stack Search algorithm increases, but it gives higher throughput quality meeting the user requirements.

For clarifying the various throughput results in the experiments through the different used web services test set sizes, we summarize them in Figure 5.1 which shows the first found solution throughput results among the different used variables in our study for the beam width size and test set size. Figure 5.1 clearly shows how using a smaller beam width size increases the throughput value while the overall line of throughput rises as we are using greater web services test set size.

We notice from Figure 5.1 that the throughput quality value decreases when decreasing the beam width from 600 to 300 for the 10000 test set size in a more clear way than for the 15000 test set size. Also, using a smaller test set of size 5000 do not show any difference in the throughput quality value by decreasing the beam width from 600 to 300. This proves that using bigger test set is indeed more efficient to apply with the Enahnced Beam Stack Search algorithm.

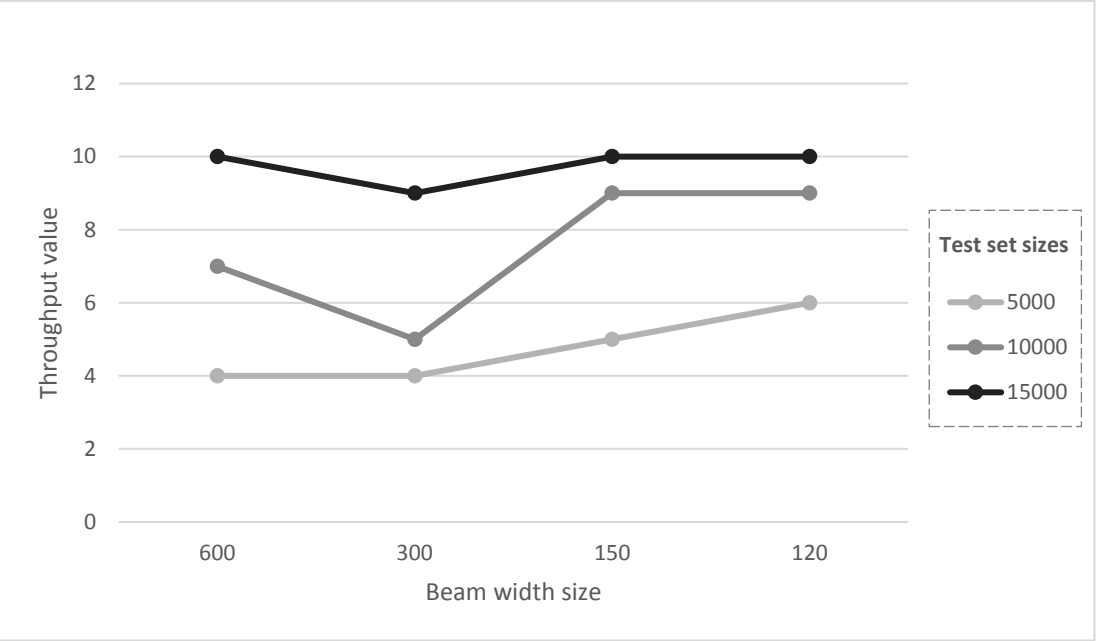


Figure (5. 1): First found solution throughput

Figure 5.2 illustrates the optimal found solution throughput workflow among the different used values of beam width size and test set size which shows that the throughput value decreases as the beam width decreases and performs better by using a bigger web services test set size.

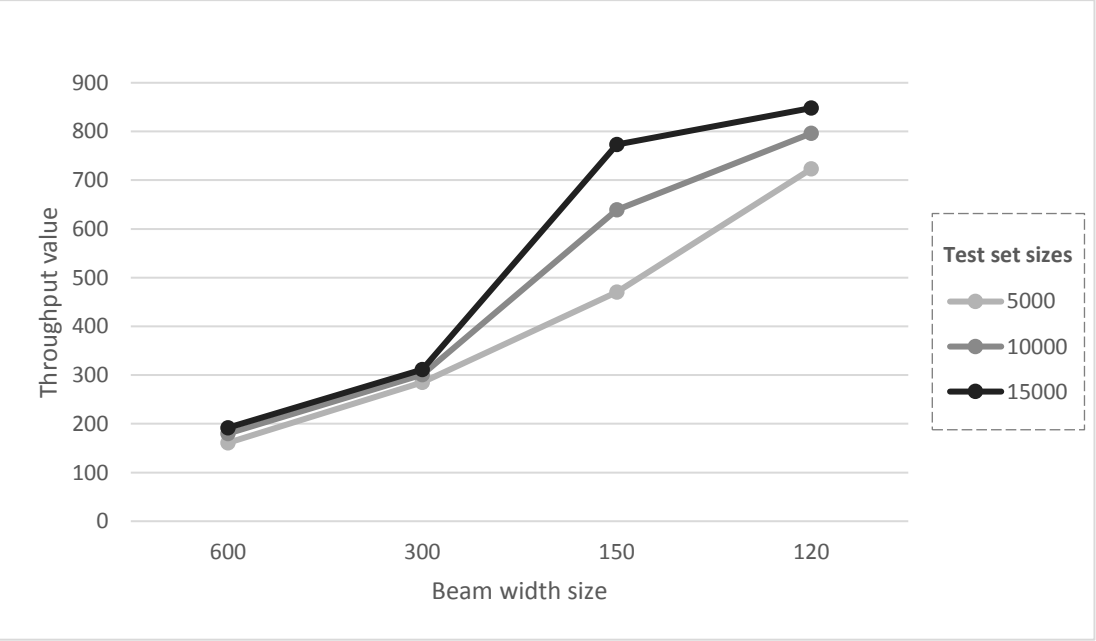


Figure (5. 2): Optimal found solution throughput

Results shown in Figures 5.1 and 5.2, show how the Enahnced Beam Stack Search algorithm performs better with bigger test set sizes and performs better using smaller beam width size.

Quality ratio values indicate the inverse relationship between the beam width and the throughput of the web services composition solution. Figure 5.3 illustrates the experiment quality ratio results using the records from the test set of size 10000 as an example from the experiments. This shows that increasing the beam width decreases the quality ratio for both the throughput and the response time which is good for the throughput in case of choosing the optimal solution and good for response time in case of choosing the first found solution.

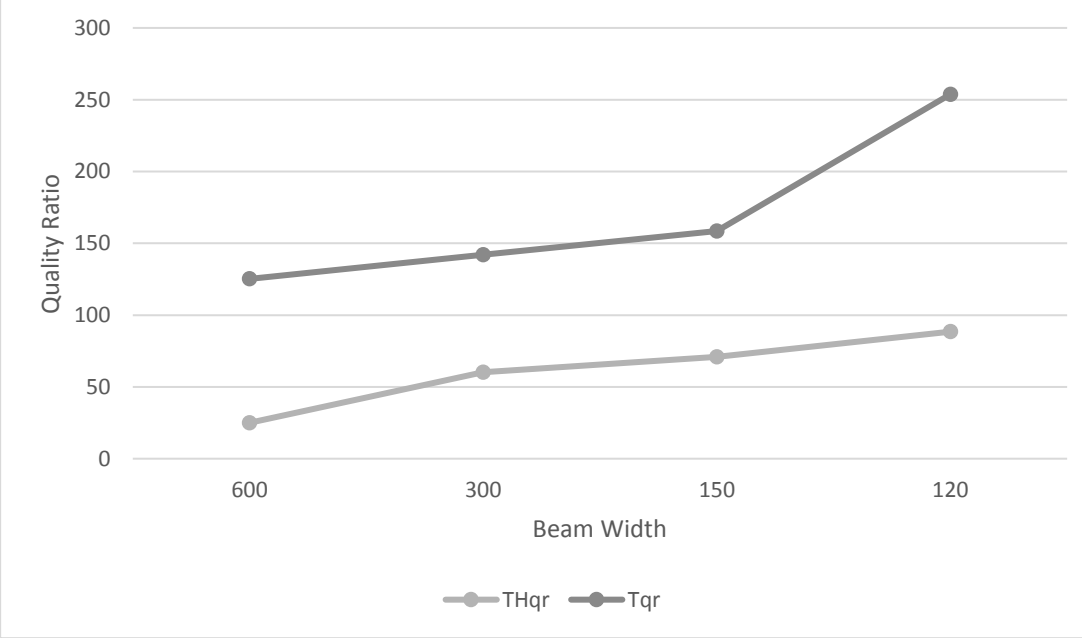


Figure (5. 3): Experimental results when the test set size 10000 results

Figure 5.3 shows the fact that decreasing the beam width decreases the complexity of the Enahnced Beam Stack Search algorithm because it gives better throughput quality regardless of increasing the response time.

Increasing the beam width decreases the chance of finding the optimal solution. For example, going back to Figure 4.2 and thinking as S_{3c} can be reached by S_{2a} and S_{2c} and they have different qualities and let's say that S_{2c} has better quality but reaching S_{3c} by S_{2c} does not give the optimal solution. Now the path through S_{2a} which holds

the optimal solution is ignored because of preferring S_{2c} at an early stage. So, choosing large beam width leads to ignoring some paths which hold the optimal solution for the user. While decreasing the beam width, allows the Enahnced Beam Stack Search algorithm to discover the highest number of possible paths for solving the web services composition problem.

The results show the importance of the Enhanced Beam Stack Search algorithm in finding the required web service that meets non-functional requirements of the user's request. As it clarifies the defference between using bigger test set size and smaller one, besides showing the defference between using smaller beam width and bigger one. Also, this enhancement finds the optimal solution inorder to reach the user’s request.

5.4. Summary

Based on the results of the expermintes, a higher difference between the first found solution and the optimal solution means that we have a high throughput quality ratio. The larger is the throughput quality ratio, the better experimental results. Quality ratio values indicate the inverse relationship between the beam width and the throughput of the web services composition solution. The throughput value decreases as the beam width decreases and performs better by using a bigger web services test set size.

Decreasing the beam width decreases the complexity of the Enahnced Beam Stack Search algorithm because it gives better throughput quality regardless of increasing the response time. Decreasing the beam width allows the Enahnced Beam Stack Search algorithm to discover the highest number of possible paths for solving the web services composition problem. Using bigger test set is indeed more efficient to apply with the Enhanced Beam Stack Search algorithm.

Chapter 6
Conclusions and Recommendations

Chapter 6

Conclusions and Recommendations

Through this work, we have studied the problem of web services composition with quality constraints in particular the response time and the throughput. The Enhanced Beam Stack Search algorithm is used to solve the web services composition problem.

It iterates over all of the web services, formed as a search graph, to discover the possible solutions and each time it finds a new solution with better throughput quality. The Enahnced Beam Stack Search algorithm is flexible, where the user can terminate it after discovering the first solution to get the fastest one and it allows the user to terminate it anytime and to choose the reached solution. On the other hand, when it continues searching it performs filtering for the found solutions to return the best one at the end.

We used 5000, 10000, and 15000 web services test set files with their associated syntactic (WSDL) and semantic (WSLA) descriptions, as we applied the experiment on them with four different beam widths 120, 150, 300, and 600.

The results show that increasing the beam width decreases the throughput of the optimal solution which is not preferred while decreasing the beam width increases the throughput. Also, if the user is satisfied with the first found solution, smaller beam width performs faster in returning the first found solution. At the same time the Enahnced Beam Stack Search algorithm performs better with bigger test set size.

The quality ratio for the throughput increases while the beam width decreases and the response time quality ratio has a reverse relationship with the throughput quality ratio. This is not considered a weakness in our approach because the optimal found solution throughput results deserve waiting for more time especially for users who can compromise optimal solution with time.

Therefore, the Beam Stack Search results a quality solution for the composition problem and decreasing the beam width in the Enahnced Beam Stack Search algorithm contributes in decreasing its complexity and discovers a better solution for the web services composition problem that meet the requirements of the user request.

Our work is the only study that uses an enhancement of the Beam Stack Search algorithm depending on finding the optimal solution and uses different test set sizes with different beam widths to solve the web services composition problem and study the results of finding the required web service that meets non-functional requirements of the user's request.

The Enhanced Beam Stack Search algorithm solves the composition problem depending on decreasing the complexity of the search space through the different levels by decreasing the beam width. But there is a limitation when a node is expanded and reaches a solution, it is pruned and cannot be reached by a previous level again since a simple operator algorithm is used in solving the problem. We suggest a future work to solve this in the algorithm by making a combination between the Enhanced Beam Stack Search algorithm and the Genetic algorithm. This is to combine the benefit of decreasing the complexity through the limited beam width with the advantage of Genetic algorithm to use complex operators in the search process. Also, we propose to apply the enhanced Beam Stack Search algorithm with a wider search space and study the efficiency of the algorithm and the quality of the solution. This can be combined with extra different beam width sizes.

References

Aine, S., Chakrabarti, P. P., & Kumar, R. (2007). A window constrained anytime heuristic search algorithm. *IJCAI*, 2250-2255.

Albreshne, A., & Pasquier, J. (2010). *Semantic based semi-automatic web service composition.* Switzerland: computer Department.

Albreshne, A., Fuhrer, P., & Pasquier, J. (2009). *Web services orchestration and composition.*

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). Web services. In G. Alonso, F. Casati, H. Kuno, & V. Machiraju, *Web Services* (pp. 123-149). Springer Berlin Heidelberg.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S. (2003). Business process execution language for web services.

Arkin, A., Askary, S., Fordi, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S. (2002). Web service choreography interface (WSCI) 1.0. W3C.

Austin, D., Barbir, A., Ferris, C., & Garg, S. (2004). Web services architecture requirements. *W3C Working Group Notes*, *22*.

Barry, D. (2017). *Service architecture*. Retrieved June 20, 2017, from www.service-architecture.com: http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html

Bartalos, P., & Bieliková, M. (2012). Automatic dynamic web service composition: A survey and problem formalization. *Computing and Informatics, 30*(4), 793-827.

Blake, B. M., Weise, T., & Bleul, S. (2010). Wsc-2010: Web services composition and evaluation. *Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference* (pp. 1-4). Perth, WA, Australia: IEEE.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004, Febreuary 11). *Web services architecture*. Retrieved June 20, 2017, from W3C Working Group Note: https://www.w3.org/TR/ws-arch/

Chan, P. P., & Lyu, M. R. (2008). Dynamic web service composition: A new approach in building reliable web service. *22nd International Conference on Advanced Information Networking and Applications* (pp. 20-25). Aina: IEEE.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). Web services description language (WSDL). *1*(1).

Oasis Committees (2016). *OASIS UDDI Specification TC*. Retrieved June 20, 2017, from oasis-open.org: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec

Cotfas, L. A., Diosteanu, A., & Smeureanu, I. (2010). Fractal web service composition framework. *Communications (COMM), 2010 8th International Conference* (pp. 405-408). IEEE.

Decker, K., Sycara, K., & Williamson, M. (1997, August). Middle-agents for the internet. *In IJCAI, 1*, pp. 578-583.

Doshi, P., Vembu, N., & Zhao, H. (2011, May 24). *Web service composition*. Retrieved June 20, 2017, from http://thinc.cs.uga.edu/: http://thinc.cs.uga.edu/thinclabwiki/index.php/Web_Service_Composition

Dumas, M., & Wohed , P. (2011). *BPEL evaluation results*. Retrieved June 20, 2017, from Workflow Patterns Initiative: http://www.workflowpatterns.com/evaluations/standard/bpel.php

El Kholy, M., & Elfatatry, A. (2015). Intelligent broker a knowledge based approach for semantic web services discovery. *Evaluation of Novel Approaches to Software Engineering (ENASE), 2015 International Conference* (pp. 39-44). IEEE.

Feier, C., Polleres, A., Dumitru, R., Domingue, J., Stollberg, M., & Fensel, D. (2005). Towards intelligent web services: The web service modeling ontology (WSMO).

Garofalakis, J., Panagis, Y., Sakkopoulos, E., & Tsakalidis, A. (2006). Contemporary web service discovery mechanisms. *Journal of Web Engineering, 5*(3), 265-290.

Hu, Y., & Wang, H. (2008). Constraints in web services composition. *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing* (pp. 1-4). IEEE.

Hunter, J., & McLaughlin, B. (2015). *Class SAXBuilder*. Retrieved June 20, 2017, from http://www.jdom.org/: http://www.jdom.org/docs/apidocs/org/jdom2/input/SAXBuilder.html

Hunter, J., & McLaughlin, B. (2000). *JDOM*. Retrieved June 20, 2017, from www.java2s.com: http://jdom.org/

JOpera. (2016). *JOpera for Eclipse*. Retrieved June 20, 2017, from jopera: http://www.jopera.org/

Kil, H., & Nam, W. (2013). Efficient anytime algorithm for large-scale QoS-aware web service composition. *International Journal of Web and Grid Services, 9*(1), 82-106.

Likhachev, M., Gordon, G. J., & Thrun, S. (2003). ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*.

Marshall, D. (2016). *Heuristic search*. Retrieved June 20, 2017, from users.cs.cf.ac.uk: http://users.cs.cf.ac.uk/Dave.Marshall/AI2/node23.html

McIlraith, S., & Son, T. C. (2002). Adapting Golog for composition of semantic web Services. *KR, 2*, 482-493.

Medjahed, B., Bouguettaya, A., & Elmagarmid, A. K. (2003). Composing web services on the semantic web. *The VLDB Journal—The International Journal on Very Large Data Bases, 12*(4), 333-351.

Mirbel, I., & Crescenzo, P. (2010). From end-user's requirements to web services retrieval: A semantic and intention-driven approach. *International Conference on Exploring Services Science* (pp. 30-44). Springer Berlin Heidelberg.

Moghaddam, M., & Davis, J. G. (2014). Service selection in web service composition: A comparative review of existing approaches. *Web Services Foundations* (pp. 321-346). Springer New York.

NetBeans.org. (2016). *Archived NetBeans IDE documentation, NetBeans 6.1 SOA docs archive*. Retrieved June 20, 2017, from NetBeans: https://netbeans.org/kb/archive/

Oh, S.-C., Lee, D., & Kumara, S. R. (2006). A comparative illustration of AI planning-based web services composition. *ACM SIGecom Exchanges, 5*(5), 1-10.

O'Keefe, R., & Costa, V. S. (2015, December 10). *Graph algorithms*. Retrieved June 20, 2017, from http://www.softpanorama.org/: http://www.softpanorama.org/Algorithms/graph_algorithms.shtml

Paikari, E., Livani, E., & Moshirpour, M. (2011). Multi-Agent system for semantic web service composition. *International Conference on Knowledge Science, Engineering and Management* (pp. 305-317). Springer Berlin Heidelberg.

Qi, S., Tang, X., & Chen, D. (2012). An automated web services composition system based on service classification and AI planning. *Cloud and Green Computing (CGC), 2012 Second International Conference* (pp. 537-540). IEEE.

Richter, S., Thayer, J. T., & Ruml, W. (2010). The joy of forgetting: Faster anytime search via restarting. *International Conference on Automated Planning and Scheduling (ICAPS)*, (pp. 137-144).

Seo, Y.-J., Jeong, H.-Y., & Song, Y.-J. (2005). Best web service selection based on the decision making between QoS criteria of service. *International Conference on Embedded Software and Systems* (pp. 408-419). Springer Berlin Heidelberg.

Shehu, U., Epiphaniou, G., & Safdar, G. A. (2014). A survey of QoS-aware web service composition techniques. *International Journal of Computer Applications*.

Sheshagiri, M., DesJardins, M., & Finin, T. (2003). A planner for composing services described in DAML-S. *International Conference on Automated Planning and Scheduling (ICAPS) 2003 Workshop on planning for web services.*

Sivasubramanian, S. P., Ilavarasan, E., & Vadivelou, G. (2009). Dynamic web service composition: Challenges and techniques. *Intelligent Agent & Multi-Agent Systems, 2009. IAMA 2009* (pp. 1-8). International Conference on. IEEE.

Srour, A. I., Othman, Z. A., & Hamdan, A. (2013). An automatic web services composition model using Ant-Colony system. *International Journal of Innovation, Management and Technology, 4*(4), 435-438.

Sycara, K., Klusch, M., Widoff, S., & Lu, J. (1999). Dynamic service matchmaking among agents in open information environments. *COMPUTER SCIENCE PUBLICATIONS, 28*(1), pp. 47-53. Retrieved June 20, 2017, from http://scholar.uwindsor.ca/computersciencepub/6

Talantikite, H. N., Aissani, D., & Boudjlida, N. (2009). Semantic annotations for web services discovery and composition. *Computer Standards & Interfaces, 31*(6), 1108-1117.

UDDI Consortium. (2001). Uddi executive white paper.

Vadlamudi, S. G., Aine, S., & Chakrabarti, P. P. (2011). A memory-bounded anytime heuristic-search algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 41*(3), 725-735.

Wang, R., Guttula, C., Panahiazar, M., Yousaf, H., Miller, J. A., Kraemer, E. T., & Kissinger, J. C. (2011). Web service composition using service suggestions. *2011 IEEE World Congress on Services* (pp. 482-489). IEEE.

Wang, Y., & Vassileva, J. (2007, June). A review on trust and reputation for web service selection. *Distributed Computing Systems Workshops ICDCSW'07. 27th International Conference* (pp. 25-25). IEEE.

Wikimedia Foundation, I. (2016, December 6). *Acme packet*. Retrieved June 20, 2017, from wikipedia: https://en.wikipedia.org/wiki/Acme_Packet

Wikimedia Foundation, I. (2017, January). *Beam search*. Retrieved June 20, 2017, from www.wikipedia.org: https://en.wikipedia.org/wiki/Beam_search

Wu, D., Parsia, B., Sirin, E., Hendler, J., & Nau, D. (2003). Automating DAML-S web services composition using SHOP2. *International Semantic Web Conference* (pp. 195-210). Springer Berlin Heidelberg.

Yan, H., Zhijian, W., & Guiming, L. (2010). A novel semantic web service composition algorithm based on QoS ontology. *Computer and Communication Technologies in Agriculture Engineering (CCTAE), 2010 International Conference. 2*, pp. 166-168. IEEE.

Yan, K., Xue, G., & Yao, S.-w. (2009). An optimization ant colony algorithm for composition of semantic web services. *Computational Intelligence and Industrial Applications, 2009. PACIIA 2009. Asia-Pacific Conference. 2*, pp. 262-265. IEEE.

Yu, T., Zhang, Y., & LIN, K.-J. (2007, May). Efficient algorithms for web Services selection with end-to-end QoS constraints. *ACM Transactions on the Web (TWEB), 1*(1), 6.

Zhang, R., Arpinar, I. B., & Aleman-Meza, B. (2003). Automatic composition of semantic web services. *ICWS, 3*, 38-41.

Zhou, R., & Hansen, E. A. (2005). Beam-Stack search: Integrating backtracking with Beam search. *International Conference on Automated Planning and Scheduling (ICAPS)*, (pp. 90-98).