1-1-2009

Web service composition ranking based on QoS and social network analysis

Alireza Dehghani
Ryerson University

WEB SERVICE COMPOSITION RANKING BASED ON QOS AND SOCIAL NETWORK ANALYSIS

By

Alireza Dehghani

B.Eng. in Computer Engineering (Software), Azad University South Tehran, Iran, May 2003

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

In the program of

Computer Science

Toronto, Ontario, Canada, 2009

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signature

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature

Web Service Composition Ranking Based on QoS and Social Network Analysis

Master of Science, 2009

Alireza Dehghani

Computer Science

Ryerson University

ABSTRACT

Web service composition refers to the aggregation of web services for producing composite solutions in order to satisfy user requirements which can't be satisfied by atomic services. It is an essential challenge to find the most reliable and trustable complex services in each composition process. Many of current composition approaches use QoS (Quality of Service) values to select among different composition candidates. However, QoS values can't be trusted all the time since some service providers may promote their services by publishing wrong QoS values. Social network analysis techniques such as PageRank have been used successfully in finding the trustable and authoritative web resources. We believe that these techniques can also be used to improve the service composition process. We have developed a modified PageRank algorithm called Service Rank in order to find the importance level of each service in a composition based on its connectivity and invocation history. This can be accomplished by assigning higher weights to links which have more number of invocations, more up-to-date invocation time and contract signing time, and longer contract durations. Eventually the Service Rank score will be combined with the QoS score for composition ranking. Preliminary results from our experiments have proved the effectiveness of this method. As a consequence users can be more satisfied with the service composition result.

ACKNOWLEDGEMENTS

This thesis would not have been possible without various people providing me the support and help required to finish my work.

First of all, I would like to express my extreme gratitude to my supervisor Dr. Cherie Ding for her continuous support, helpful guidance and discussions during the development of this thesis. I am also greatly thankful for her encouragement, assistance, valuable time, and precise editing of my thesis.

I would like to express my sincere gratitude and deep appreciation to Dr. Abdolreza Abhari, Dr. Alexander Ferworn, and Dr. Isaac Woungang for their very valuable comments.

I extend my thanks to all professors I have come to know in the computer science department for their academic support and guidance during the past two years.

My hearty thanks also go to my parents for their understanding, support, utmost solicitude and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background and Motivations

Services can be any kind of functionality, which are capable of being invoked through a particular interface [45]. In the SOA (Service Oriented Architecture) model, services are self-contained software units of functionality. Several services can be put together to create a composite service which offers an overall functionality. This procedure is called service composition. Service composition assists us to take advantage of presently existing web services to offer a completely new service. Composition is helpful if we are searching for a web service with particular inputs and outputs, and there is no existing web service to give us the desirable results whereas combination of a few services could offer us good results [19].

Combined atomic web services or web service compositions are considered as self-describing, self-contained modular applications which are capable of being located, invoked and published on the web in a way similar to atomic web services [19]. Currently many organizations and companies only develop and host their core business applications and outsource other application services. Therefore, the capability of effectively and professionally choosing and integrating inter-organizational and heterogeneous services on the web at runtime is a very important task for service composition [43]. The problem that all the service composition techniques are trying to solve is to rank a set of web services composition solutions which if the best one is executed in a particular setting, offers the optimized requested service for the service consumer [19].

Numerous automatic service composition algorithms have been proposed which employ AI planning or other techniques, and their objective is to automate different compositions steps [7] [19] [28] [45] [60].

The process of automatic service composition can be accomplished in these steps [43]:

1- **Presentation of single service:** Service providers present their atomic services by employing some available languages for advertising. The fundamental attributes to portray a web service consist of the signature, the states, and the non-functional values. The signature is signified by the service's inputs, outputs and exceptions, which offers information about the data transformation throughout the execution of a web service and is important for building the web service I/O (Input/Output) graph. Preconditions and post-conditions specify the states. Modeling is based on the transformation from one set of states to another. Attributes which are employed for evaluating the services, for instance cost, security, and other quality properties are non-functional values which will be needed for QoS analysis.

2- **Translation of the languages**: The external and internal service specification languages are distinguished in most service composition systems. For enhancing accessibility, users employ the external languages in a manner which assists them to express what they need and are able to provide. External languages are different from the internal ones which are employed by the composition process generator. The reason is that the process generator needs more specific and formal languages (e.g. the Logical Programming Languages). WSDL (Web Services Description Language) and DAML-S (DARPA - Defense Advanced Research Projects Agency

Agent Markup Language for Services) are standard web service languages which are common and usually employed as external languages. It is a good idea to develop the translation components between the standard web service languages and the internal languages.

3- **Generation of composition process model:** There is a service specification language for the service requester to express the requirement. This requirement can then be specified by a process generator by composing the atomic services advertised by the service providers. The process generator considers the functionalities of services as input, and output process model which portrays the composite service. A set of chosen atomic services, the control flow and data flow between these services are contained in the process model (I/O graph).

4- **Evaluation of composite service [40] [43]**: Usually there are many services that have similar functionalities, which causes the generator to make more than one composite service to meet the requirement. In this situation the composite services can be evaluated on their overall usefulness by employing the information provided from the non-functional attributes and other criteria. The most usually employed data is the non-functional attributes. Relative importance of every non-functional attributes could be further specified by the requester. Usage history data of used compositions can be another useful source of information for the evaluation. This history data can help decide which link in the I/O graph is more important than others. In our work, social network analysis algorithms will be used to exploit the history data, which is then combined with the QoS data, in order to achieve higher

trust-ability in highly ranked composite services. The composite service with the highest combined rank is the best one.

5- **Execution of composite service:** After choosing a specific composite process and presenting the composite service, it is time to execute the composite service. Composite service can be considered as a chain of message passing in agreement with the process model. The action which is the transition of the output data of a previously executed service to the input of a later executed atomic service is called data flow of the composite service [43].

As explained before, when there are more than one composition solutions, how to select among them will be difficult for users. Due to this reason, ranking is a crucial part of every automatic service composition procedure, which is also the main focus of our work.

1.2 Problem Statement and the Proposed Approach

One popular way of finding composition solutions is using web service I/O graphs. In a graph structured web service composition and ranking, a web service is represented as a node in the graph, and its inputs and outputs are connected through links [3]. If the output of service *A* is the input of service *B*, there is a link from *A* to *B*. Given the required input and output, by traversing the graph, all candidate composition solutions could be found. Furthermore, semantic information such as goals and service properties can be used to reduce the size of this set. Even so, it is still difficult for users to go through the list and make their selection. There are many research works on how to find the composition solutions, but not many on proposing a good ranking algorithm to help users in this selection process. It is our goal of this study to propose such a ranking algorithm.

Because of the exponentially increasing search space in composition problems, it is critically important to design a good ranking algorithm. The main advantage of the graph based technique is that it eliminates some limitations of the AI planning techniques. This goal is possible by proposing dynamic and incremental graph based composition and ranking techniques. These methods facilitate composition in environments that have parties which are subject to unpredicted changes [30].

Most of the research works on service composition ranking or selection use the QoS data to make the decision. However, QoS data alone can't guarantee the trust-ability of the solution because of the possible false or misleading QoS data published by service providers (e.g. to promote their services). Also, QoS parameters are run-time related and their values may change over time [49]. Therefore, we should look for a ranking algorithm which can be employed in a dynamic environment and can guarantee the trust-ability.

We often can trust monitoring agents, but because of their high cost, they are not included in many registries and we need some other ways to guarantee the trust-ability. Social network analysis which is a suitable method for dynamic problems can be a great option for finding trustable solutions by looking at previous users' invocation history [30].

In this thesis we propose to combine both QoS evaluation and social network analysis techniques, in order to find trustable and high quality composition solutions. We will consider two data sources: QoS of individual component services, and the service rank calculated based on the service I/O graph and the invocation history log. QoS data is still important especially for new services without any prior invocation history.

There are many different definitions on trust-ability. In this thesis, trust-ability means that a service requestor can trust a composition solution with high confidence. We will use this definition throughout this thesis. Our main idea is that if many requestors have paid some fee to invoke a service, it usually means this service has a high quality, which is not based on the QoS values published by providers themselves, but is based on the real usage data. We want to use the usage history to enforce the trust-ability of the composition solution.

1.3 Web Services Composition Ranking

Different methodologies can be employed for ranking web service compositions. Efficiency is a very important factor to be considered. While a composition problem in general has exponentially increasing search space, it is essential to thoughtfully decide about selecting primary search algorithms and data structures for ranking final composition solutions. The wise choice is the ranking algorithm which is based on social network analysis [30]. The interpretation behind this option is that link analysis has been successfully applied to problems, in which, partial evaluation of the search space was required, either due to the huge size of the search space itself or the shortage of extensive information [15] [63].

As mentioned before, while an atomic web service can't satisfy a customer's request, the relation between the input and output parameters of atomic web services, could be employed to compose web services in order to satisfy different customers' demands dynamically. After discovering the web services and finding the I/O graph based on the relation between the input and output parameters of atomic web services, we could start to generate different composition solutions and rank them. This goal can be accomplished by employing link analysis techniques such as a modified PageRank algorithm combined with the QoS data.

Because we don't deal with web pages, we use the name Service Rank Score (SRS) instead of PageRank score. Since many of the publishers can't be fully trusted, employing PageRank algorithm can increase trust-ability of the selected compositions. Combining Service Rank Score with QoS data could give us more trustable results.

The feasibility of this solution could be justified by the fact that a social network for web services can be generated in the service registry [54], where all the service description information is available and the history logs could also be saved. The current implementation of UDDI (Universal Description, Discovery and Integration) registry doesn't have this capability of saving QoS information or history logs. But we believe that by extending UDDI, it is possible to achieve this goal, if we could setup an intermediary between the requestor and the provider and submit all the traffic data to the registry.

Based on the input and output parameters of services, we first build the service I/O graph, then history logs assist us to weigh different links in the I/O graph, and finally we run our Service Rank algorithm and QoS based ranking algorithm. In the experiment part, we will compare both techniques in separate and combined manners to observe whether these techniques are able to attain a more effective ranking result by calculating an average rank for user selected compositions. By achieving a higher average rank, we can demonstrate that our approach can improve the ranking for those user selected compositions.

The main contributions of this thesis are: 1) proposing a novel social network analysis approach of ranking the composition solutions based on the connectivity and invocation history of component services; 2) modifying the PageRank algorithm by including the usage-based weights for all the links; 3) categorizing a quite complete list of QoS attributes and

quantifying the way of aggregating the QoS values of component services; 4) developing a combined approach of ranking compositions based on both QoS and social network analysis approach we proposed.

1.4 Thesis Outline

In this chapter we have introduced the basics of service composition and ranking, discussed the problems we want to solve, and explained our proposed solutions briefly. The rest of the thesis will be organized as follows:

In chapter 2, we will survey various link analysis methods and the state of the art of current works which employ graph structures and other techniques to assist incrementally and dynamically compose web services and construct a new service that has desirable inputs and outputs which satisfies the user requirements. We will also briefly survey the basics of link analysis and QoS analysis.

In chapter 3, we will define how we calculate the Service Rank score, how to calculate the overall QoS score for compositions, and how to combine them together to achieve more reliable results. Also, the creation of history logs which assist us to weigh different links in our I/O graph will be elaborated in this chapter.

In chapter 4, we will compare both techniques in separate and combined manners to observe how these techniques are able to attain a more effective ranking result for different solutions. We describe an implementation of our methodology, present the simulation steps, and connect these steps with outcome back to users and designers of the procedure. This chapter also considers the practical ways that our method can be applied to rank and compose

web services through a number of case studies. The outcomes of these case studies are employed in different evaluations and discussions in the result analysis part.

Finally, Chapter 5 offers a conclusion on our work carried out, an outlook of projected future efforts and final remarks with considering how the contribution of this thesis assists in other fields of associated research works and the way further research projects will develop and in which courses this might obtain better achievements.

CHAPTER 2

LITERATURE REVIEW

2.1 Service Composition Methods

As mentioned before service composition helps us to take advantage of existing web services to offer a completely new service [19]. Service ranking and selection is essential in creating composite services. While a designer builds up a composite service, where several services are supplied by outside partners, it should be able to deal with the difficulties related to this issue [33]. There are different methods for composition and ranking web services [8] [17] [31] [48]. Usually composition methods are QoS based, link analysis (or graph analysis) based, or combination of QoS and link analysis based methods. In this chapter, different techniques for composition and ranking will be explained.

Composition of web services requires finding service providers which are able to satisfy users' functional and nonfunctional demands [46]. Nonfunctional demands usually refer to QoS requirements. QoS attributes are parameters which have to be quantified properly to offer more optimized web services composition ranking results. By considering proper measurement process and description for QoS metrics [23] it is possible to propose service composition and ranking techniques which are based on QoS attributes. Each measurement process for a QoS metric must include some accepted characteristics. QoS metrics will be explained, categorized, and quantified further in details later in next chapter. As mentioned before, the disadvantage of purely QoS-based composition ranking techniques is a possible

lower trust-ability of the final composition solutions, considering that QoS data might be manipulated by service providers.

Link Analysis techniques (graph based) can assist us to rank compositions dynamically in cases where QoS is not valid or reliable for evaluating composition solutions. Higher trust-ability is the most important advantage of link analysis techniques which must be considered in ranking different compositions. In this chapter we briefly survey graph based techniques for automatic service composition to have a better foundation for our work, including AND/OR Graph based service composition [29], service composition using Enhanced Service Dependency Graph [16], incremental graph-based approach for service composition [45], Planning Graph Model based service composition [60], and more other techniques. We will also briefly survey the basics of link analysis.

There are some techniques for composition and ranking which employ combined techniques. The advantage of combination is to have advantages of both methods and eliminate their disadvantages. Combing QoS and graph based techniques assists us to have a higher reliability with the capability of dynamic service composition approach [22] [61]. In this thesis, a new technique belonging to this category will be proposed.

2.2 Service Composition and Ranking Using QoS Analysis

With an increasing number of web services presenting equivalent functionalities, the non-functional attributes of a web service such as QoS attributes have become crucial selection criteria and critical for service providers to satisfy customers' requirements [34].

A Web service ws_i can be illustrated by the pair (FS_i, QS_i), where FS_i is to represent the service's functionality, QS_i is a description of its QoS attributes. WSDL is usually used for

expressing a web service's functional characteristics, and there is no common way for describing a service's QoS parameters [9]. These QoS attributes (non-functional properties) such as throughput, security, availability, and response time have to be quantified properly in order to have a more efficient web service composition ranking and selection procedure [9][10].

There should be a proper measurement process and description for QoS metrics to offer service providers and consumers a common perspective [62]. For instance, response time is one of the QoS metrics which is depending on the workload intensity level, and a single value is not suitable for representing it [35]. Therefore a better way for measuring response time is to consider its average value over a period of time. The measurement process for each QoS metric should have some accepted characteristics. These characteristics include what and how the metrics should be measured, who should accomplish this task and in which place of the network it is intended to be measured.

One important reason for employing QoS-based service composition (e.g. in [47]) is to have a guaranteed quality for delivered services in the final composition. As mentioned before the dynamic web service composition [39] brings in challenges to current composition techniques. Some techniques attempt to integrate QoS analysis into service composition. The problem of this method is that the composition which is based on the QoS values proposed in advance which might be manipulated and can't be trusted in real cases. To solve this problem, in [52], small test cases are run, to identify real QoS values and therefore assist with service composition and ranking. Other parts such as service execution engine, audit system, selection algorithms, etc. can be comprehended by employing QoS technique with small

adaptations. The solution offered in this work is just a research proposal, and the belief is that the realization of this proposal is able to assist solving the problem of guaranteed QoS level. The main problem of this work is not considering the dynamic nature of the composition and ranking processes in case of having problems with test cases due to hardware failure or other issues.

In [10] a method is proposed to support the design process to have composite services supplemented with quality descriptions which can be further employed to define more composite service qualities. This method assists to have a better description and evaluation of composite service qualities. To achieve the goal a mathematical model is offered with a double range. First, this model is capable of describing a series of processes related to the required qualities in one set. It is required that actual services have to be chosen before the implementation of abstract services. Second, this model tries to resolve the problem of process selection by using the class of customers with their quality requirements and quality weights established by the customers. The goal of this method is to allow companies, providers, and developers to have a common perspective, which is the basis for potential improvements. Improving the ability to describe and evaluate services' qualities will let the different service providers and customers to understand each other so as to create more advanced tools to upload more aspects of service design including definition, design, discovery, monitoring, selection, and contracts.

WS-BPEL (Business Process Execution Language) is a language for illustrating the actions of a business process which is based on the relations between the processes and their associate processes. In [38], authors propose an advanced QoS calculator model for WS-

BPEL processes. This QoS determiner offers values for response time, cost, and reliability for every activity in a WS-BPEL model. Also, a QoS tool is proposed which allows the designer to change components of the BPEL process workflow, for instance, inserting fault tolerance constructs, and verify any QoS improvements in the process. This model has some problems such as not supporting discovery of exclusiveness of links, not handling isolated scopes (it allows to manage concurrent access for shared resources), and not capturing forced termination actions because of inability to work with termination handlers.

Service selection is a crucial part of every QoS based service composition. In [6], the broker proposes a service selection procedure for composing web services that supports various QoS classes which is totally different from the majority of the offered plans for service selection. Also, authors in [6] consider SLAs (service level agreement) surrounding the stream of requests and try to resolve and formulate the optimization problem by employing linear programming method. SLA is an efficient technique to estimate the quality of different services. It offers a method to catch the attention of more customers for different service providers, and constructs a type of novel business association through creation of the assurances at particular service level. This includes offering the reimbursement in case of inability for offering the guaranteed service level. SLA is able to assure multiple service levels and fast service composing and modifications in SOA models [58].

The proposed approach in [6] is able to control the service selection in an actual working broker-based architecture. The proposed problem formulation is capable of being changed to consider extra QoS attributes or to modify an offered SLA. The proposed model also offers a statistical assurance on the required QoS attributes.

The authors of [55] studied complex service composition problem with multiple QoS constraints. A broker-based framework (QCWS) is proposed for QoS aware Web service composition. In this framework, web service composition with QoS guarantee contains two steps. The first step is service planning and the second step is service selection. The first step is accomplished by the Selection Manager (SM) and the second step is done by Composition Manager (CM) in the QoS broker. In addition, complex service‘s service selection problem with single QoS requirement is analyzed and studied. Also, the system model is expanded to be able to deal with multiple QoS requirements by employing two different models. The first model is defining the problem as a multi-dimension, multi-choice setback (the combinatorial model), and the second is describing the problem as an optimal path problem (the graph model) [55]. The goal of service selection is maximizing a user-defined utility function under the different QoS constraints. The definition of utility function might contain an extensive set of system parameters such as client QoS requirement (QoS constraint), static server information (service level), and network factor to reach some user specific goals. To evaluate their performances, simulation outcomes on using different service selection algorithms (optimal and heuristic) to choose and compose services under several QoS constraints are presented.

In [57], the basis for discovering the qualitative characteristic of web services composition is studied. This facilitates the construction of a system being able to provide an end to end advanced QoS to the customer by employing both the RS (Region Switching) algorithm and WS-Notification. The authors changed the level of monitoring from passive to active (e.g. observing information content instead of verifying web services availability

which is usually accomplished by other systems). This improvement gives considerable advantages since other systems are deeply relying on real-time data. As a consequence, this service provider may propose outstanding quantifiable QoS metrics. Some users may decide not to employ their services in the outlook, since the qualifiable QoS aspect was lost. To solve this problem, the RS algorithm which could control a complete multi web services composition process is employed. RS algorithm forms the composition process into a FSM (Finite State Machine) model, constructed above of the WS Notification infrastructure that offers denotations for message delivery. As of a web services composition call, the system sends notices to all participating services through the WS-Notification standard. This will activate announcements to be directed to every registered client when a modification happens inside a web service. Before obtaining an announcement message, the RS algorithm recognizes the web service influenced by this announcement, and estimates how this influence will change other web services. After that, it will publish a request to re-calculate a possible subset of the whole process planning to return an important update to the client. On top of it, an enhanced Decision Support Systems (DSS) can be constructed to create improved optimization scenarios and analyses. The problem of this method is how to incorporate this system with current QoS aware brokers intended for web services evaluation and composition. In [5], authors propose a heuristic based method to resolve the problem of QoS aware Web Service composition. They offer a heuristic algorithm (called H1_IP_RELAX) that employs a backtracking method on the outcomes calculated by a program. The evaluation of the results concludes that this heuristic algorithm is quick and provides outcomes which are close to the optimized solution.

In [2] authors studied the problem of QoS-based Web services composition in presence of service dependencies and conflicts, by offering a GA (Genetic Algorithm). They offered Genetic Algorithm because it is scalable [20] and the experimental outcomes have proven that the computation time of the GA algorithm rises linearly while the quantity of the web services increases, and which the computation time is not exaggerated by the quantity of Web services. Therefore, the offered algorithm is able to be employed for big dynamic environments [57] with a huge number of Web services. The proposed GA is extensible, and also, the proposed algorithm considers five non-functional properties, which can be simply expanded to contain more nonfunctional properties. This algorithm looks effective and for most of the tested problems, the proposed GA can discover a reasonable solution in a short period of time.

2.3 Graph Based Service Composition and Ranking

In this section we survey graph-based techniques for web service composition. First we will briefly review the social network analysis concept in order to have a better background for the later reviews.

2.3.1 Overview of Link Analysis Methods

Link Analysis is an unsupervised method which is not just a single algorithm but is referring to a collection of algorithms which are bounded with the data on which they operate [25] [32]. This type of data should be able to be represented in the form of nodes and links. Nodes (vertices) represent entities that can be people, documents, etc. Links (edges) indicate the relationships between nodes. Link Analysis can be used in a wide range of applications such as viral marketing, law enforcement applications, epidemiology, information retrieval, etc

[11] [21] [24] [44] [53]. The broad goal of link analysis is to employ algorithms to analyze the networks and acquire new relational information from the networks.

Link analysis algorithms for search engines continue to grow in importance due to the complexity of Internet by considering a variety of new users' demands which are not satisfied by the traditional algorithms [26] [41]. Old search techniques focused on the search keywords but new search engine techniques are focusing on the relationship among pages. Therefore link analysis, which its goal is to find relational information from the networks, is suitable to be employed in new search techniques. Because we are going to employ link analysis techniques for ranking composite web services it is better to study them before employing them in order to have a better understanding for their usage.

Nowadays, Internet users expect relevant, reliable, and authoritative results from search engines. Authoritative pages are pages which are linked to by many other pages and recommended by many people. Therefore authoritative pages always have some important and reliable content related to the search topic. This is similar to the research papers which are cited by many other papers. For instance, if a user is looking for political news, CNN page is an authoritative page for her or him to click. The reason is that many news websites link to CNN as the source of the news. It is quite common that in the search results we can find many web pages which are not related to search topics. The aim of the PageRank algorithm is to find a strategy to distinguish these authoritative web pages from junk pages. This can be done by assigning a score to every page *u* by employing this formula (PageRank algorithm) [32]:

$$R(u) = \sum_{v \in B_u} \frac{R(v)}{N_v} + E(u) \quad \textbf{(Equation 1)}$$

The notions are defined as below:

- u : A web page
- R(u) : Score (rank) of a web page *u*
- F_u : Set of pages that are pointed to by u
- B_u : Pages which point to u
- N_u : the number of links from u
- E(u) : A Priori score, e.g. it is a priori score for authoritative pages, and also, can counterbalance the effects of sinks which are pages that are pointed to but they don't point to other pages (dead links) by negative scores, because usually authoritative pages point to many other pages but dead links don't

The top search results are the relevant (based on the search keyword) and authoritative pages (based on the PagesRank scores). Google employs a combination of PageRank algorithm with other standard algorithms [41]. In our work we are looking for more accessible, reliable, and trustable nodes (which are more invoked than other nodes based on history data) in our service I/O graphs to assign higher weights to them in order to rank composition solutions. So we take a modified version of PageRank.

Kleinberg's HITS algorithm [1] also scores pages based on links. This algorithm distinguishes hubs (pages that point to many other pages) and authorities (pages that are pointed to by many other pages) in its processing steps. Constructing a sub web graph is the first step to start a search on the Internet. Sub-graph is a small network within Internet which

is rich in relevant pages on the search topics. It is possible to obtain results which do not contain the main search keywords. For instance, if the search keyword is “DVD player” it is possible to obtain Sony’s home page even though it may not contain the words “DVD player”. The second step is calculating the hub score and authority score until reaching the equilibrium. Then the results will be ranked based on their hub/authority scores separately or in a combined way [1] [32].

These two explained algorithms are based on the relation among the pages within the Internet, which demonstrates the usage of link analysis in new search engine techniques and possibly web services composition. HITS Algorithm is less efficient because it is query dependent. PageRank algorithm can be pre-computed before the actual query is submitted. We will use PageRank in our research because of this feature.

2.3.2 Graph based Service Composition Methods

In this section current works which employed graph based techniques will be reviewed. Graph based methodologies were selected because they are able to effectively attain web service composition goals. The purpose is to permit the web service composition procedure to produce suitable solutions for a composition call whenever partial evaluation of the composition network is necessary. In the graph based technique, the composition network is represented as a graph network, and then graph network analysis metrics are employed to dynamically examine graph link composition to direct solutions in a way to satisfy the composition request.

In [15], web services communications are symbolized as a graph network, where web services are symbolized by nodes, and the links among them are corresponding to edges.

Heuristic graph search algorithm is employed for the web service composer implementation, which is expressed by a series of graph network analysis ranks (such as PageRank) which dynamically inspect the relevance and importance level of every web service. Also, in the experiment part they attempt to find what metrics are more capable of satisfying the request, and which are capable of directing to the maximum composition solution performance.

Instead of evaluating the performance for every metric in a separate manner, it is possible to evaluate the average performance for every categorized ranking metric, such as local (when only the local network information is required), global (when knowledge outside a node's local region is required for the computation), absolute (when the rank's range is general, and not based on the current request), and relative (when the rank value is evaluated related to the current request). Also, these ranking categories could be combined to two levels of categorizations which are Relative-Local, Relative-Global, Absolute-Global, and Absolute-Local. The outcomes indicate that in order to achieve higher performance levels the relative and global ranking metrics perform better than the absolute and local ones respectively.

There are some problems related to this work. First, the proposed method is incomplete and it requires additional experimental efforts to extend and verify the conclusions and outcomes. Secondly, an important extension is necessary, to develop a more general assessment procedure for the composition effectiveness with several metrics (metrics which web service composer is able to employ), and also, other factors such as computational cost of different ranking metrics has to be taken into account. The third problem is that it does not have different experimental settings within different types of composition graphs, to evaluate

the performance of the offered ranking categories and metrics. The experiment which has been accomplished in this work is just for uniform link distribution which has to be extended for different link distributions and link density levels in future works. In our thesis the experiment has been accomplished for a variety of link distributions.

The technique proposed in [29] formalizes the web service composition as a search problem in an AND/OR graph. An AND/OR graph is a structure which is usually employed in automatic problem solving. Given a particular service request which can only be satisfied by a composition of web services, we need to recognize the service categories which are related to the request and then dynamically construct an AND/OR graph to capture the input/output dependencies between the web services of these service categories. The graph change is accomplished based on the conditions in the request by using the logical operators. After that, the search algorithm is employed to search the changed AND/OR graph for a complete and minimal composite service template which satisfies the service request. The algorithm can be applied repetitively to the graph to explore for other templates until the result is accepted by the service requester.

In [16], the graph illustrating input/output dependencies between service operations is called Service Dependency Graph (SDG). Since relationships in an SDG are not directly described by data models from service interface definitions, the expressiveness and reusability of the relationship are not guaranteed. There is an improved version of SDG, namely SDG+, which has clear dependency declaration by conveying dependencies straightforwardly with static explicit declarations. After clarifying the attribute dependency and operation dependency, some issues in SDG could be solved by SDG+. Because SDG+ is

an extension of SDG, same search algorithm can be applied to SDG+. After that an automatic service composition algorithm could be developed based on SDG+.

In [45], a novel approach of automatic service composition is proposed. The matchmaking and discovery is based on semantic annotations of service properties, for instance their goals, inputs and outputs. This approach employs a graph-based search algorithm to resolve all potential composition candidates. Reasoning and filtering are done by using goal-based expressions on composition candidates. Two major components have been suggested to implement this - a validation wizard and an automatic composition engine.

The work in [45] is concentrating on the study of the scalability and performance metrics of the offered graph-based search algorithm. Concerning non-functional properties in these semantic annotations, this technique is restricted to the static properties such as cost. A simple accumulation can be done on such static properties. More sophisticated aggregation models could be considered to expand to more dynamic non-functional properties such as response time.

The technique proposed in [60] is to solve the matching problem where the output parameters of a web service can be considered as the input parameters of another web service. This work targets to eliminate the redundant web services included in the planning graph.

In [4], the web service composition is again considered as a planning problem. Besides employing the semantic information throughout the search, this approach also considers non-functional attributes of the services (e.g. service quality).

In [19], web services composition is based on the graph theory with using a modeling language which is called “interface automata”. In addition, some network analysis approaches have been proposed in other works which are suitable for semi-dynamic variation of composition solution models [3]. Also, some graph based algorithms for web service composition have been presented with different data structures such as chain data structure [28].

Sometimes, we might require to record extra information about the web services in the dependency graph. As a consequence, researchers may need to benefit from other languages, for instance, Resource Description Framework (RDF) language that supplies data regarding web resources. In [19], the OWL-S (Web Ontology Language) description language is chosen to specify the behavioral properties of web services. This language identifies a web service in terms of execution workflow and its graphs, inputs and outputs.

Many of the surveyed techniques are not able to rank the final composition solutions. Many of them also did not consider the trust-ability and the variety of the QoS properties. In our work, we will try to improve the ranking system in order to find more trustable and high quality composition solutions.

2.3.3 Combination of QoS and Link Analysis Methods for Service Composition

Combination of link analysis with some ranking procedures based on QoS will give us more reliable results. Composition method proposed in [33] falls into this category. A quality of service control can be performed based on the service failure rates. A local service registry contains the binding information which is updated dynamically consistent with the evaluation provided by web service users, and also the probability of having low quality

solutions with high initial ranking scores. Such binding information can be employed to estimate a network of services, and then link analysis can be applied to rank services consistent with their popularity. The quality of the final composition solution can be evaluated by performing a simulation experiment. The experiment outcomes demonstrate that by considering the failures that consumers experienced, this method notably performs better in choosing relevant services. One of the problems of this method is that employing link analysis (e.g. PageRank) on the binding repository might be expensive in case of a huge network size. The experiment in this work has no control over the percentage of employing both quality control and social network analysis to offer better solutions. In this thesis we try to overcome these obstacles by offering a combined score based on both QoS and service rank score which let one to find the best combination of both factors for the ranking procedure. Also, the service rank metric defined in this thesis is based on both PageRank score and user feedbacks with their invocation dates.

A ranking method offered in [51] combines a few QoS properties with social analysis calculation to increase the performance of highly ranked solutions. Services are capable of creating a social network through invocation associations. In this method the composition solutions are sorted not only by considering their QoS values from the beginning of the procedure but also their popularities in terms of services' usages by different clients and services. By this combination, the average rank of a user selected compositions will increase if it has a better performance.

The social ranking feature of the proposed algorithm is the frequency of the service usages by other services. A request from a client to a server is considered as a rating. The client

evaluates all the requests to compute a local rating of the server. Global ranking computation is based on the aggregation of local ratings. Number of service clients, frequency of services' invocations by other services, and QoS (in this work availability, response time and in one experiment throughput) for each service, are three considered factors for the aggregation. The ranking is performed in two levels. The first level is local ranking which considers frequency of requests for each service, and QoS attributes. Also, local rankings have to be normalized to remove the effects of bogus services which send artificial requests to particular services in order to promote them. Global ranking values are driven from the normalized local rating matrix. Authors of [51] also try to prevent distributing global ranks to bogus services if they are not pointed to by reliable services, which could prevent malicious services from obtaining higher ranks.

At the end some experiments have been conducted on real-world data to verify the goals. The results showed different performance from different services with same functionality in different day times. The second experiment studies how QoS impacts the ranks of the services. Results show that by increasing the invocation numbers of each service we will have lower performance of them which decreases their ratings. Therefore traffic has to be directed in a proper way to have less overloading on services to prevent deterioration of services' rankings. In the third experiment the effect of monitoring on the procedure has been studied. The outcomes of the third experiment demonstrate that monitoring has little effect on increasing the processing time. The fourth experiment demonstrates that it is capable of being employed for thousands of services in the real environment.

In method proposed in [51] QoS attributes have been combined with social analysis methods, and these two methods are indistinguishable in the work. Also, this method does not consider that having unreliable QoS data in the beginning will make the ranking untrustworthy. In this thesis we employ PageRank for ranking which is totally different from the social ranking technique proposed in [51]. The reason is that the main contribution in this thesis is employing the history log to improve trust-ability rather than just focusing on non-functional (QoS) attribute values of the solutions from the beginning. Also, in this thesis many other QoS attributes have been considered, studied, defined and combined with the defined service rank.

2.4 Conclusion

As mentioned above, several services can be composed to construct a composite service which produces an overall functionality. One method of composition is using QoS-based techniques. The reason for employing QoS-based service composition is having a guaranteed quality for delivered composition services. Graph based methods use I/O graphs in order to offer an overall functionality to requesters. These graphs are constructed by using the semantic inputs and outputs of different web services.

Unfortunately, many graph based algorithms do not have an efficient strategy to rank composition solutions. Our work is a combination of these two methods in a novel way (with employment of history logs) which is intended to find a trustable technique for ranking composition solutions. In order to have the most valid and high quality solution at the end, a variety of QoS attributes should be defined, quantified and categorized. By combination of these QoS attributes with the modified well known link analysis methods, our method can

improve pervious techniques to find high quality and at the same time reliable composition solutions.

CHAPTER 3

METHODOLOGY

When an atomic web service can't satisfy a user's requirement, the I/O graph (which is constructed based on the semantic inputs and outputs of web services) should be employed to find composite services. The objective is to satisfy functional and non-functional users' requirements dynamically. After discovering the related atomic web services we should start to generate different composition solutions and then find the optimal composition based on the I/O graph. This goal can be accomplished by employing social network analysis techniques such as the modified PageRank algorithm combined with QoS information. Because we don't deal with web pages we define it as Service Rank Score (SRS) instead of PageRank score.

A social network for web services based on past invocation data can be generated in the service registry, and then the Service Rank algorithm could be used and combined with QoS attribute values for selecting the best composition solution. In the rest of this chapter we will define different methodologies to achieve our goals.

3.1 Overview of the Algorithm

Our proposed algorithm for the composite web services ranking and selection begins with a service graph. The network of web services constructs a graph, in which each service is considered as a node, and the relations among the nodes are considered as edges. On other side usage history log records the information related to the past service invocations.

We can define a set of parameters for each web service including input parameters and output parameters of the service [60]. When a web service is invoked with its input set it will return the output set. It can be defined that a web service *A* has a link to service *B* if input set of B is a subset of Output set of A.

In fact, a web service network is a network consisting of a set of nodes and a set of links. Each link connects a pair of nodes (e.g. A to B) showing a connection from a web service to another web service.

As was explained before, each node is a web service, and each link demonstrates the relation between web services. For instance, Figure 1 illustrates a web service graph in which service A links to B and C, B links to C, and C does not point to any nodes. Other details of this figure will be explained in later sections.

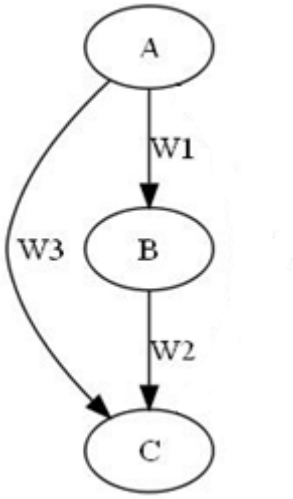


Figure 1 - A sample web services graph

We assume that all of the composition solutions can be found based on the service I/O graph by using some existing algorithms (for instance, a tool has been developed for this

thesis which is able to find all the routes between two nodes in an I/O graph). Afterwards the composite services are required to be ranked, because usually there are more than one solution being found based on the I/O matching due to the numerous possible routes between two nodes in a given I/O graph.

Depending on how this set of solutions is going to be ranked, there are two main categories of approaches of our interest. One approach is evaluating a composite web service based on QoS factors. The other is using social network analysis algorithm to rank all of the web services independent of QoS factors.

PageRank algorithm, which is one of the well known link analysis algorithms, can be employed and modified to calculate a ranking score for all web services in the attained I/O graph based on how they were connected with each other and how frequently they were invoked before, which is called Service Rank algorithm in our work. A service can be ranked high by Service Rank algorithm if it is invoked by many other web services, or invokes many other web services which are semantically connected to them. Also, for calculating the service rank score we should not ignore weights in our I/O graph which are attained from history logs. Calculating weights from history logs will be explained in details in the next section.

Because we need to rank the different composition solutions by employing the Service Rank algorithm, a service rank unique value for a composition solution should be defined based on its components' service rank values and weights of the links. Having higher service rank scores for compositions means that they have higher probability to be chosen and

invoked by the requesters, and also, their component services are usually more reliable and trustable.

On the other hand, in order to satisfy requestors' QoS requirements, QoS data also needs to be considered during the ranking process; to achieve this goal we should propose a way to evaluate QoS parameters for getting an overall quality score for a composite web service. This goal can be achieved by categorizing and quantifying different QoS parameters and defining a unique value for overall QoS level. The last step is to compare this value with the requester's QoS demanded value, and combine it with the service rank unique score, to rank the solutions.

Based on what was explained before, here we employ both approaches (QoS and Service Rank) to increase trust-ability of selected compositions. Before starting the ranking process stated above we have two main steps. The first is finding (or generating) a composition I/O graph which contains web services related to a request (with their I/O relationships) and the second step is considering the available web service descriptions and their semantic input and outputs in this I/O graph to generate our solutions. Then, for each composition solution we consider the ranking scores of all nodes, based on both QoS factors and Service Rank scores (with considering links' probability weights obtained from history logs). Also, our solution can be further improved by eliminating extra links, to attain a better graph for the composition. For simplicity, in these solutions in case of having multiple links (if they point to the same side) between two web services, only one link should be kept. Our main focus in this work is on ranking and selection of available composition solutions in order to satisfy a user's functional and non-functional requirements.

3.2 I/O Graph and Weights Calculated on History Log

History logs are attained from the past service invocations by different users. These logs can be saved in service registries. It is possible to use these logs to assign weights to different links in our I/O graph, and to discover more reliable composition solutions. There are different types of relations in the I/O graph, including sequential, concurrent, conditional and selective relations [34]. If invocation of a service such as ws_2 requires invocation of another service such as ws_1 then they have relation in a sequential manner. We denote this sequential composite relation by "^" operator (e.g. ws_1^ws_2). If invocation of a service such as ws_3 requires invocation of other services such as ws_1 and ws_2 together, then invocations of ws_1 and ws_2 are concurrent and they have concurrent relation in our composite solution. We denote this concurrent composite relation by "*" operator (e.g. (ws_1 *ws_2) ^ ws_3). To show conditional relation we use '~' sign (e.g. ws_1 ~ws_2), and for selective invocation we employ '+' sign (e.g. ws_1+ ws_2). These types of invocations, their illustrations, and their calculation methods for QoS values will be discussed in details in sections 3.3.2 and 3.3.3. In this section we will explain methods to calculate different links' weights from history logs.

Three important factors have to be considered for calculating weights in I/O graph. The first one is the web service users' contracts of invoking that service, the second is the dates of invocations, and the third is to consider frequency of invocations in our history logs. The important parameters which should be evaluated in contracts are durations and start dates. This approach assigns higher weights to more frequently invoked, latest and valid web services and distinguishes them from other services.

Table 1 and Table 2 demonstrate a portion of a sample history log. A list of requests' invocation dates with their user data are displayed in Table 1, and different users' contracts data is listed in Table 2.

Table 1 - Invocations in the history log by users U_1, U_2, U_3

Requested Service	Invocation Time	User
A	22/03/2007 4:30 PM	U_1
A^B^C	22/04/2007 2:30 PM	U_1
A^B*(C^E)	22/07/2007 1:30 PM	U_1
A^B^C^D	14/12/2006 8:30 AM	U_1
A^B	11/03/2006 8:00 AM	U_1
A^B	18/05/2007 9:30 AM	U_2
B^C	15/06/2007 10:30 AM	U_2
A^(B+D)*(E^F)	17/02/2007 1:30 PM	U_2
C	21/03/2006 11:30 AM	U_3
C^(D+F)	16/05/2007 4:00 PM	U_3

Table 2 - Information from different users' contracts

Requested Service	Contract Start date and Duration	User
A	22/03/2007, 1month	U_1
A^B^C	22/03/2007, 2 months	U_1
A^B*(C^E)	22/07/2007, 4 months	U_1
A^B^C^D	14/12/2006, 6 months	U_1
A^B	10/03/2006, 3 months	U_1
A^B	18/03/2007, 4 months	U_2
B^C	15/03/2007, 1 month	U_2
A^(B+D)*(E^F)	14/02/2007, 2 months	U_2
C	20/03/2006, 3 months	U_3
C^(D+F)	17/03/2007, 18 months	U_3

The first important factor for weighting links in our I/O graph is users' contracts (e.g. SLA – Service Level Agreement). If we have contracts with earlier start dates and longer duration

time, then we will assign a higher weight for links in them. The rationale behind is that if the contract duration is longer, it usually means consumers have a higher trust level on this service. Below we first define the weight based on the start date of the contract:

Let l_{ij} be the link from l_i to l_j

Let L_{ij} be the set of all occurrences of l_{ij} in the log

Let l_{ijr} be the *r*-th occurrence of l_{ij}

Then W_{SDC} is the weight of start date of contract for l_{ij}:

$$\mathbf{W_{SDC}(l_{ij})} = \frac{1}{|L_{ij}|}\sum_{l_{ijr}\in L_{ij}} \frac{1}{\boldsymbol{current\ date - start\ date}(l_{ijr})} \qquad \textbf{(Equation 2)}$$

After that we will define the weight based on the duration of the contract - W_{DC} for l_{ij}:

$$\mathbf{W_{DC}(l_{ij})} = \frac{1}{|L_{ij}|}\sum_{l_{ijr}\in L_{ij}} \boldsymbol{Duration}(l_{ijr}) \qquad \textbf{(Equation 3)}$$

The second important factor for weighting links in I/O graph is the invocation date. The higher rank should be given to the more recently invoked links. The rationale is simple, because we prefer newer data. The weight of invocation date W_{ID} will be defined as follows:

$$\mathbf{W_{ID}(l_{ij})} = \frac{1}{|L_{ij}|}\sum_{l_{ijr}\in L_{ij}} \frac{1}{\boldsymbol{current\ time - invocation\ time}(l_{ijr})} \qquad \textbf{(Equation 4)}$$

Note that here the unit of time is minute.

The third factor to calculate weight is frequency. In table 1, it is apparent that we have four invocations of A → B by the user U_1. In order to prevent the possible spamming or putting too much weight on one user's history data, we give higher weight to the first invocation by a requester, lower weight to the second invocation by the same requester, and ignore the rest of the invocations. Therefore, for example, we don't count A→B invocations

more than two times from user U_1. We can set some predefined coefficients for weighting invocation frequencies. The following formula is to calculate the weight for invocation frequency in the history log:

Let *F1* be the frequency of the first invocations by different consumers (coefficient c_1 is set to 1),

Let *F2* be the frequency of the second invocation by different consumers (c_2 is set to 0.1),

$$\mathbf{W_F(l_{ij}) = c_1 \times F1 + c_2 \times F2} \qquad \textbf{(Equation 5)}$$

For example, we can calculate the weight for A→B as follows:

F1 (A^B) = 2 (it is invoked by two different users)

F2 (A^B) = 2

$W_F(A^B) = c_1 \times F1 + c_2 \times F2 = 1 \times 2 + 0.1 \times 2 = 2.2$

After all the weights have been calculated, Let W_I be the final invocation weight for l_{ij}, then we will have:

$$\mathbf{W_I\ (l_{ij}) = W_F(l_{ij}) \times W_{ID}(l_{ij}) \times W_{SDC}(l_{ij}) \times W_{DC}(l_{ij})} \qquad \textbf{(Equation 6)}$$

Final results will then be normalized to the range [0,1] based on the following formula, and later in this chapter this normalized weight will be referred to as W_{ij}.

$$\mathbf{NW_I\ (l_{ij}) = \frac{W_I\ (l_{ij}) - Min\ of\ (W_I\ (l_{ij}))}{Max\ of (W_I\ (l_{ij})) - Min\ of\ (W_I\ (l_{ij}))}} \qquad \textbf{(Equation 7)}$$

$$\mathbf{W_{ij} = NW_I\ (l_{ij})} \qquad \textbf{(Equation 8)}$$

In continue we will employ these weights to calculate service rank scores in our I/O graph which will be explained in later sections.

3.3 Social Network Analysis of Web Services

Social network analysis techniques have improved web search to become more accurate and trustable by employing algorithms such as PageRank or HITS. The actual usage data (e.g. click-through data [59] and access patterns [56]) can further be utilized to improve the performance. The nodes in a web graph are web pages and links in the graph are from the hyperlinks between web pages (nodes).

Web services also can form a social network graph despite of the fact that the web services' social network is not as straightforward as that of web pages. If we assume that we can save history logs of the real usage data in one or more web services registry centers, we can apply link analysis algorithms for ranking the web services as we are able to accomplish for a network of web pages. The web service graph can be obtained by generating the I/O graph and then assigning weights to all links based on history logs, Figure 1 illustrates such a graph.

In Figure 1 (a sample web services graph), there are three services A, B, and C, which are assumed to be offered by three independent providers. We presume that we have the service links as follows: 1) Service A has a link to service B with weight W_1 and to C with weight W_3(weight is the probability that service A invokes service B based on history logs); 2) Web service B has a link to Service C with a weightW_2. We treat weights as the probabilities to reach nodes in our constructed service graph which later we will employ to calculate Service Rank scores. Here we are focusing on the I/O graph and the probabilistic composition graph will be explained in later sections.

Web services network can play a significant role for web services discovery, and in particular for generating compositions and ranking them based on a request which can't be satisfied by an atomic service [30].

3.3.1 Service Rank Score for a Single Service

A process to discover the graph properties according to the link distributions among the nodes of the graph is called the link analysis of a graph. This technique has been employed for the ranking of web pages for the Internet searching. There are several link analysis ranking algorithms, such as HITS, Salsa [12] and PageRank. Since PageRank is the most popular technique in web page ranking due to Google's success, we will employ and modify it to demonstrate how a link analysis technique can be employed for ranking web services [12] [33].

A service can be ranked high by PageRank algorithm if it is pointed to by many other web services or points to many other highly ranked web services based on their semantic inputs and outputs. Here we assume that our requester is looking for a series of services in the I/O graph. In other words, PageRank algorithm here presumes that a requester can eventually reach its required demands at any link in the I/O graph (because we are going to generate different composition solutions from the I/O graph). The goal is to measure the importance level of a web service in the I/O graph by looking at its invocation histories if it is invoked by many other services or invoking many other highly ranked services, which demonstrates that the level of service importance or trust-ability is higher.

We define PageRank score for web services with the damping factor as follows: Assume that web service A points to a set of other services $FS(A)$, and A is also pointed to by a set of

services *BS* (*A*). We denote the size of *FS* (*A*) by *CS* (*A*). The PageRank score *PRS* (*A*) of the service *A* is given by:

$$PRS(A) = (1\text{-damps}) + damps \times \sum_{b \in BS(A)} \frac{PRS(b)}{CS(b)} \quad \textbf{(Equation 9)}$$

In the original PageRank algorithm the probability from one node to another is the same for the whole graph. However, based on the invocation history, these probabilities could be different. So our Service Rank formula adapted from this original PageRank formula is defined as below (b_i points to b_j):

$$SRS(\mathbf{b_j}) = (1\text{-damps}) + damps \times \sum_{i=1}^{k} \frac{SRS(b_i) \times W_{ij}}{CS(b_i)} \quad \textbf{(Equation 10)}$$

By this formula we employ the invocation weights calculated based on the history log in our Service Rank score. How to calculate the weights has been explained in the previous section.

If we use the example from Figure 1 and assume that:

$W_1 = W_2 = W_3 = 1$

Also, by initializing *SRS*(*A*), *SRS*(*B*), and *SRS*(*C*) to 1, it is possible to iteratively calculate the scores until they converge to a final result.

SRS(*A*) = 0.15 + 0 → No nodes point to A

SRS(*B*) = 0.15 + 0.85 × (*SRS*(*A*) / 2) × W_1

SRS(*C*) = 0.15 + 0.85 × (*SRS*(*A*) / 2 × W_3 + *SRS*(*B*) / 1 × W_2)

By solving this set of equations in 20 iterations, we get *SRS*(*A*) ≈ 0.15, *SRS*(*B*) ≈ 0.213750, and *SRS*(*C*) ≈ 0.395438. As we can observe from obtained values, web service C

has the highest PageRank score since it is pointed by all other nodes (more nodes point to C). After that, we will normalize the Service Rank scores by using this formula:

$$\text{SRSF}(\mathbf{S_i}) = \text{Normalized}(\mathbf{SRS}(\mathbf{S_i})) = \frac{\mathbf{SRS}(\mathbf{S_i}) - \mathbf{Min\ of}\ (\mathbf{SRS}(\mathbf{S_i}))}{\mathbf{Max\ of}(\mathbf{SRS}(\mathbf{S_i})) - \mathbf{Min\ of}\ (\mathbf{SRS}(\mathbf{S_i}))} \qquad \textbf{(Equation 11)}$$

Based on what was explained before, Service Rank scores generally reflect the relative trust-ability (how well a service in the I/O graph is reachable and invoke-able by the requester) and importance of the web services in the network.

3.3.2 Probabilistic Composition Graph and More Invocation Types

As we illustrated and mentioned before in Figure 1, each service in I/O graph is represented as a node and is connected to other nodes via directed links. This figure only shows the sequential relationship between services, and there is equal probability from one node to another. The actual composition graph could be much more complicated, for instance, in Figure 2, the probability which web service A invokes web service B is *P*1 and C is *P*2. These probabilities demonstrate that invocations are not independent and are conditional. We call this kind of invocation as the conditional invocation. This probability is different from the weight which we defined before. The probabilities are presumed to be one if no value is mentioned in the composition graph (e.g. for concurrent and selective invocations) [34].

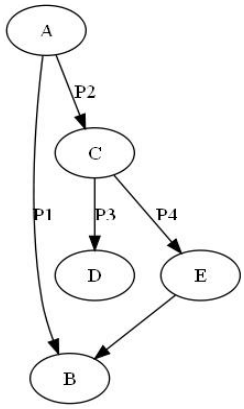


Figure 2 – More complex composition graph

In Figure 3(a) we have a sequential invocation. Sequential activation is when a web service is activated as a consequence of completing of a predecessor web service. In Figure 3(b) a web service requires its successors to be invoked in parallel and probabilities are assumed to be one, and this is called the concurrent invocation. In Figure 3(c) we have a conditional invocation with different probabilities. These invocations are not independent from each other. In the fastest-predecessor-triggered activation, as long as one of the predecessors completes, it will activate the web service as shown in Figure 3(d) and this is called the selective invocation. When a web service is activated after the completion of all of its predecessor web services, it is called the synchronized activation (join), which could be considered as another type of concurrent invocation, as in Figure 3(e). In synchronized activation, all of the probabilities have to be one, and it can't be conditional [34].

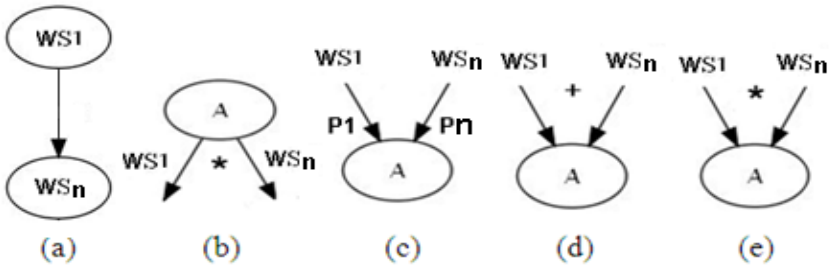


Figure 3 [34] – Illustration of more complete list of invocation types

We can use the composition graph in Figure 4 to explain these different types of invocations. In Figure 4(a), service F is activated when either B is completed (it should be after A invokes B) or after D is completed (after A invokes C and C invokes D). In Figure 4(b) web service A requires its successors to be invoked in parallel and probabilities should be one. And web service F is invoked after the completion of all its predecessors D and B, which are good examples for synchronized activation (join).

(a) **(b)**

Figure 4 – Different types of invocations in complex compositions

3.3.3 Service Rank Unique Value for the Composite Service

Because Service Rank can be considered as importance probability to invoke a web service in an I/O graph, we will use the following probability rules [36] to calculate the *Service Rank Unique Value (SUV)* for concurrent compositions:

Rule 1: Complement Rule

If A and A' are complements → P(A) + P(A') = 1 **(Equation 12)**

Rule 2: Addition Rule

P(A OR B) = P(A) + P(B) - P(A AND B) **(Equation 13)**

Rule 3: Mutually Exclusive Rule

If A and B are mutually exclusive then P(A AND B) = 0

P(A OR B) = P(A) + P(B)-P(A A AND B) **(Equation 14)**

Rule 4: Multiplication Rule

P(A AND B) = P(B) P(A|B) **(Equation 15)**

P(A AND B) = P(A) P(B|A) **(Equation 16)**

Rule 5: Independence Rule

If A and B are independent, then:

* P(A|B) = P(A) (a) **(Equation 17)**

* P(B|A)= P(B) (b) **(Equation 18)**

* P(A AND B) = P(A) P(B) (c) **(Equation 19)**

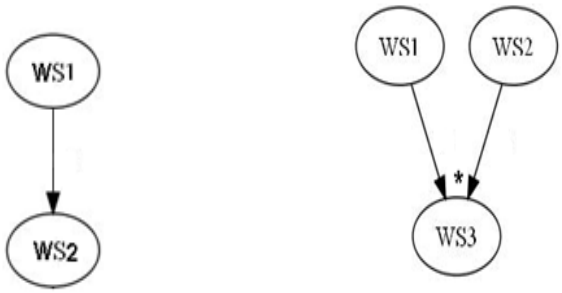
* P(A OR B) = P(A) + P(B) - P(A)P(B) (d) **(Equation 20)**

Service Rank Unique Value (SUV) will be defined based on different types of invocations in a generated composition solution. Service Rank score of a single service demonstrates how well it is reachable by a requester in the I/O graph and how frequently it is actually invoked by different requestors, which in a way measures how trustable it is to make this selection. Service Rank score of the composite service measures the overall trust-ability of the composition solution. For instance, if we have a sequential invocation CS_1: $ws_1 \rightarrow ws_2$ as in Figure 5(a), then we can define its SUV score using this formula:

$$SUV(CS_1) = \frac{(\mathbf{SRS(ws1)} + \mathbf{SRS(ws2)})}{\boldsymbol{LF(CS_1)}} \qquad \textbf{(Equation 21)}$$

LF is length of CS_1 which is the number of its nodes. The reason for defining this kind of length factor is to avoid selecting long chain of services. The compositions with shorter length will be preferred. In general to calculate the SUV score for a sequential invocation we will have this formula:

$$SUV(CS) = \mathbf{LF(CS)^{-1}} \times \sum_{\mathbf{ws_i} \in \mathbf{CS}} \mathbf{SRS(ws_i)} \qquad \textbf{(Equation 22)}$$



(a) (b)

Figure 5 – Sequential and concurrent web services used for SUV calculations

In above formula a composite service is denoted as CS which contains a set of services WS= $\{ws_1, ws_2, \ldots\}$. WS demonstrates a subset of registered services from the history

graph which are employed in our generated composition solutions. These services can be composed together to satisfy the requester's demands based on their relations in our I/O graph. ws_i can be an atomic web service or another composite service.

In Figure 5(b), because ws_1 and ws_2 are concurrent services, as a consequence, we have "AND" relationship between their Service Rank scores. We can merge these probabilities using rule 5(c). In the concurrent composition which is illustrated in Figure 6, we have (n-1) concurrent invocations. Therefore, to define a SUV score for concurrent invocations with a more generalized formula, we simply calculate the P (ws_1 AND ws_2 AND … AND ws_{n-1}), which is equal to $\prod_{i=1}^{n-1} SRS_i$. SRS_1,..., SRS_{n-1} . We have this generalized formula for concurrent invocations (the formula for the opposite direction invocation will be same):

SUV(CS)= $\prod_{i=1}^{n-1} \mathbf{SRS}(\mathbf{ws_i}) + \mathbf{SRS}(\mathbf{ws_n})$ **(Equation 23)**

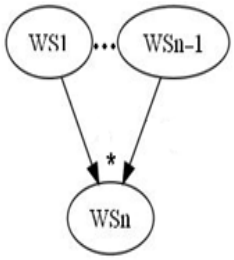


Figure 6 – Concurrent invocations

If invocation of a service such as ws_0 is followed by the invocation of only one of the services such as ws_1 or ws_2, then invocations of ws_1 and ws_2 are conditional and they have conditional relation in our composite solution.

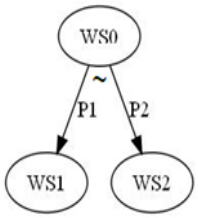


Figure 7 – Conditional invocations

Because ws_1 and ws_2 are conditional services, we have "OR" relationship between SRS1 and SRS2, and for each invocation we also have a probability value (Figure 7). In the selective composition which is illustrated in Figure 8, we have n selective invocations. Therefore, to define a SUV score for the selective invocation with a more generalized formula, we simply calculate P(ws_1 OR ws_2 OR … OR ws_n) for selective invocations. Note that we should also consider probabilities P1,…, Pn-1 for different selections. We have this generalized formula for selective invocations (Figure 8) which is same as the formula for conditional invocations (Figure 7):

SUV(CS)=$\sum_{i=1}^{n}(\mathbf{P_i} \times \mathbf{SRS}(\mathbf{ws_i}))$ **(Equation 24)**

In Figure 3(d) we have selective relations among predecessor services. We have this generalized formula for this type of composition as follow:

SUV(CS)=$\boldsymbol{max}_{i\in 1\sim n}(\mathbf{SRS}(\mathbf{ws_i}))$ **(Equation 25)**

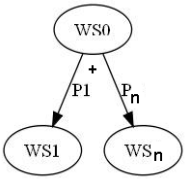
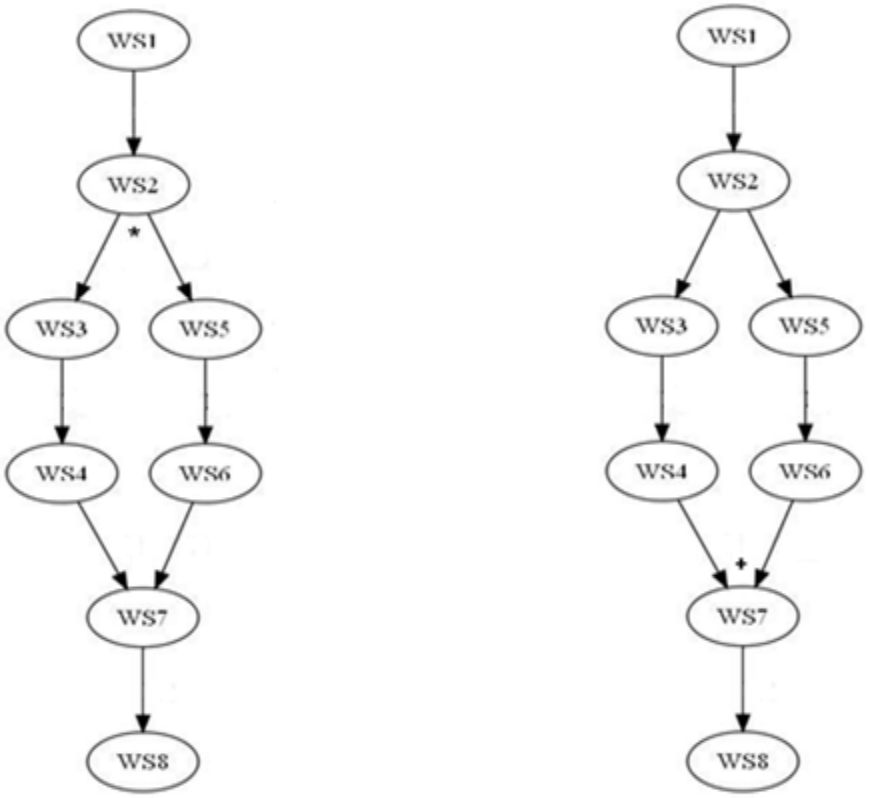


Figure 8 - Selective invocations

More complicated examples are shown in Figures 9(a) and 9(b) (probabilities are assumed to be one) in which we have one sequential invocation in the beginning, a concurrent invocation in the middle of Figure 9(a) and a selective invocation in the middle of Figure 9(b), and finally one sequential invocation at the end. It is important to mention that it is impossible to combine Figures 9(a) and 9(b), since if we have a concurrent invocation of branches in the middle of them, it will be impossible to have selective invocations of the same branches.



(a) (b)

Figure 9 - Composition samples

We calculate the SUV for the composition in Figure 9(a) by combining previously defined formulas:

SUV(CS)= 3^{-1}×(SUV(ws_1) + SUV (ws2^ws3^ws4) × SUV(ws2^ws5^ws6)+ SUV(ws_7^ws_8))

Also, we calculate the SUV for the composition in Figure 9(b) as follows:

SUV(CS)= 3^{-1}×(SUV(ws_1)+max(SUV (ws2^ws3^ws4) *and* SUV(ws2^ws5^ws6))+ SUV(ws_7^ws_8))

As we can see, different parts of the compositions have sequential or concurrent relationships to each other; therefore we combined their SUV calculations in order to get the final SUV for the solution.

These formulas will give us a unique value for the composite solution's service rank unique score. A higher SUV value indicates a better accessible and trustable composition solution. Also, if SUV values are calculated from PageRank scores instead of Service Rank scores it is called PageRank unique value (PUV). If a composition has a higher PUV score it has a better connectivity between its nodes.

3.4 QoS Categorization and Quantification

There are numerous QoS attributes related to web services. We will arrange them into different QoS categories by considering different types of requirements and quantifications. Each category has a set of measurable parameters. We will demonstrate which kind of different QoS requirements we will consider and how we can quantify them.

Here, categorization is mainly based on QoS data types. For instance, if we have Boolean values for some QoS factors, then we will categorize them into Boolean category. Or, if we

have probabilistic QoS factors we can categorize them into types such as probabilistic category. Also, we will try to have separate categories for positive parameters (the higher the value, the better) such as reliability from negative parameters (the lower the value, the better) such as cost.

For each category we will define a formula which can be used to measure and combine its QoS values to rank the composite service. At the end we will define a unique QoS value for the whole composite service.

3.4.1 Positive Numeric QoS Parameters

Positive Numeric QoS Parameters have no specific ranges and we require higher values for them to have better QoS. Runtime related QoS parameters are the major elements of this category.

Scalability is the ability to boost the computing power of the service provider's computer system [42].It indicates that the system is capable of performing how many more transactions per second. This is related to throughput and performance. If we have a higher average scalability of the composite services' nodes, we have a higher scalability for the whole composite service.

Throughput is the number of completed service requests per time period which in fact is related to latency and capacity.

Capacity indicates how many concurrent connections a service supports to have in order to have a required performance which is different from scalability. As mentioned before scalability is the capability of boosting the computing power of the service provider's computer.

To calculate throughput, capacity, and scalability values for a composite service we should consider the worst case which will be the minimum of services' throughput, capacity and scalability values.

Accuracy indicates how many errors a service makes over a period of time, e.g. in one second.

Stability/change cycle is a measure about how stable the service is and indicates how often it transforms its interface and implementation. This can be calculated by dividing its number of transformations divided by the period of time.

To calculate the accuracy and stability/change cycle values of a composite service we can simply calculate the addition of services' accuracy and stability/change cycle values. The reason is that the number of errors and transformations has to be considered for every web service in the composition.

We normalize their values to a range between 0 and 1 in order to have a standard value for the simulation, comparison and evaluation for each composition.

Positive Numeric QoS Unique Value (NPQUV) is the combination of positive numeric QoS parameters that is defined as the addition of Scalability, Accuracy, Stability/change cycle, Throughput, Completeness, and Capacity.

3.4.2 Negative Numeric QoS Parameters

Negative Numeric QoS Parameters have no specific ranges and if they have lower values is better. Performance is the major part of this category which is to measure the speed of serving a request.

Latency and Response time: *Response time* is the guaranteed time needed to complete a service request. *Latency* is the required time between the service request received and the request is being answered which can be considered as the average delay on serving a request [42]. In general to calculate the latency for a sequential invocation (e.g. Figure 4), we will have this formula:

Let *LT* be the total latency of a composite web service *CS* which consists of *n* services, and let lt_i be the latency of the component web service *I*, then we have:

$$LT(CS)=\sum_{i=1}^{n}(\boldsymbol{lt_i}) \quad \textbf{(Equation 26)}$$

In the concurrent composition which is illustrated in Figure 6, we have (n-1) concurrent invocations. Therefore, to define a total latency for concurrent invocations with a more generalized formula, we simply consider the maximum latency of concurrent invocations. The maximum part is because of the fact that all predecessors should be completed. We have this generalized formula for concurrent invocations:

$$LT(CS)= MAX_i(\mathbf{lt_i})+ \mathbf{lt_n} \quad \textbf{(Equation 27)}$$

In the selective composition which is illustrated in Figure 8, we could simply consider the minimum latency. We have this generalized formula for selective invocations:

$$LT(CS)= MIN_i(\mathbf{lt_i})+ \mathbf{lt_n} \quad \textbf{(Equation 28)}$$

We can use two examples to show the calculation steps. In the first example, we calculate the total latency for the composition in Figure 9 (a) by employing this formula:

LT(CS)=LT(WS1) +MAX(LT(WS2^WS3^WS4), LT(WS2^WS5^WS6))+LT(WS7^WS8)

$$LT(CS)= lt_1 +MAX((lt_2 + lt_3 + lt_4), (lt_2 + lt_5 + lt_6))+(lt_7 + lt_8)$$

In the second example, we calculate the total latency for the composition in Figure 9 (b) by employing this formula:

LT(CS)=LT(WS1) +MIN(LT(WS2^WS3^WS4), LT(WS2^WS5^WS6))+LT(WS7^WS8)

LT(CS)= lt_1 +MIN($(lt_2 + lt_3 + lt_4), (lt_2 + lt_5 + lt_6)$)+$(lt_7 + lt_8)$

As we can observe that different parts of the composition have sequential and concurrent relationships to each other, we employ different types of latency formulas in order to calculate the final total latency for the composition solution. We normalize LT value to the range between 0 and 1. The calculation for the response time will follow the same steps.

Cost is the measurement of the cost for requesting the service and its execution. For instance we can measure the cost per request or per volume of data. Usually cost has a direct relation with web service's quality. Reliable, faster, more secure web services usually cost more. Let c_i be the cost of a web service execution. Let *C* be the total cost of the composite web service then we have:

For sequential invocation (as in Figure 5(a)):

C(CS)=$\sum_{i=1}^{n} c_i$ **(Equation 29)**

For concurrent invocation (Figure 6):

C(CS)= $\sum_{i=1}^{n-1} c_i + \mathbf{c_n}$ **(Equation 30)**

For selective composition (Figure 8):

C(CS)=$MAX_i(c_i) + c_0$ **(Equation 31)**

For conditional composition (e.g. as in Figure 7) the formula is defined as follows:

C(CS)=$MAX_i(\mathbf{p_i} \times c_i) + c_0$ **(Equation 32)**

We use the maximum value because for the cost we must always be prepared for the highest one in our composition solutions. We normalize this value between 0 and 1 to have standard values for the simulation, comparison and evaluation.

Negative Numeric QoS Unique Value (NNQUV) is the combination of latency, response time (L/R), and cost which is the addition of their inverse values (x^{-1}).

3.4.3 Probabilistic QoS Parameters

Probabilistic QoS Parameters can be treated similarly to probabilities, which makes us able to employ probability rules to define final formulas for them. These parameters are reliability, availability, and Robustness/ Flexibility which are all positive QoS factors.

Reliability is the capacity of a service to complete its essential functions successfully under acknowledged conditions for a specific period of time [42]. Therefore, we simply define it as probability of successful invocations of a service over a period of time (between zero and one). As a consequence we are able to use the probability formulas for calculating the total reliability of a composite service.

In Figure 4 (sequential invocation), the reliability of any service is very essential for our composition's total reliability. Therefore based on the probability rules we should multiply reliability of each service to get the composite service's reliability. To calculate the reliability for a sequential invocation we will use this formula:

Let r_i be the reliability of a web service in the composition;

Let *R* be the total reliability of the composite web service, then we have:

$$R(CS)=\prod_{i=1}^{n} r_i \qquad \textbf{(Equation 33)}$$

In Figure 6 we have concurrent invocation of services and service WS_n only commences when all active predecessors become completed. Because reliability is the probability of successful invocations of a service over a period of time, we can simply use AND operator to calculate total reliability here. Therefore, the formula for total reliability in case of having concurrent invocations will be defined same as sequential invocations because:

$$R(CS)=(\prod_{i=1}^{n-1} \boldsymbol{r_i}) \times \mathbf{r_n} \rightarrow R(CS)=\prod_{i=1}^{n} \boldsymbol{r_i} \quad \textbf{(Equation 34)}$$

In the selective composition which is illustrated in Figure 8, to define reliability we simply use OR operator for selective invocations. The reason is that one of the branches is needed for starting WS_n. We have this generalized formula for selective invocations:

$$R(CS)= MIN_i(\mathbf{r_i}) \quad \textbf{(Equation 35)}$$

This formula is defined because in selective invocations we should consider the worst possible case for probabilistic attributes such as reliability.

Availability is the probability which service is available and it is related to reliability. We define measurement formula as follows [27]:

Let TT be the total time.

Let AT be the time that a specific service is available.

Availability will be defined by the following formula:

$$\text{Availability}= \frac{\boldsymbol{AT}}{\boldsymbol{TT}} \quad \textbf{(Equation 36)}$$

By this formula we calculate the probability which a service is available during the total invocation time for the composition solution. Because availability is based on the probability

therefore to calculate the availability of a composite service we can simply use the probability formulas as we did for reliability.

Robustness/ Flexibility can be defined as the probability to which a service is able to function properly in case of having incomplete, conflicting or invalid inputs. Because Robustness/ Flexibility is a probability value, to calculate the *Robustness/ Flexibility (RF)* of a composite service we can simply use the probability formulas as we did for reliability.

Probabilistic QOS Unique Value (PQUV) is defined as the combination of Reliability, Availability, and Robustness/ Flexibility and its value is the addition of these QoS attributes.

3.4.4 Boolean Type QoS Parameters

Boolean QoS parameters are either zero or one. Security and Configuration Management related QoS values are the important elements of this category.

Integrity of the data, on which transactions operate, can be guaranteed by grouping transactions into separate units. The unit will be successful if every transaction in the unit "commit" or all "roll back" to their unique state in case of having a transaction failure [42]. We can assign a value 0 or 1 which indicates if we have a successful unit of transactions and guarantees the integrity of data and helps increasing the degree of transaction support by the web service.

Exception Handling indicates if the service can perform properly when a service receives less number of parameters than it needs. Here we can quantify this by assigning a value of 1 or 0 which shows if the service is able to handle the exceptions or not.

Regulatory score indicates if the service is properly aligned with regulations or not (one or zero).

Supported Standard score shows if a service complies with standards such as industry specific standards by assigning one or zero.

Guaranteed messaging requirements parameter is a value of one or zero to show if a web service guarantees the order and persistence of the messages.

Security related QoS can be defined in a more fine-grained level but because of the lack of support in current service oriented applications, we simply use the Boolean value to represent it.

Authentication indicates if the service authenticates users or other services to access it. If the authentication system exists we can assign 1 otherwise we can assign 0.

Authorization measures whether the service authorizes principals so that only they can access the secured services. If yes then we assign value 1 else 0.

For ***Confidentiality*** if the service treats the data in such a way that authorized principals are the only ones who can access or change the data then we assign value 1 to it, otherwise zero.

If the supplier is accountable for its services then the ***Accountability*** value is 1 else 0.

If the history of a service is traceable when serving a request, then we assign ***Traceability and Auditability*** score with value one, else zero.

***Data encryption* score** shows if the service encrypts data and its provider guarantees these security requirements or not. This score is either zero or one.

To calculate the final value of each of the discussed Boolean type QoS parameters for a composition, we employ the logical "AND" operator. It means calculated value is one if and only if all of the services' Boolean QoS values in composite service are one.

Boolean Unique QOS Value (BUQV) is defined as the combination of Integrity score, Exception Handling score, Regulatory score, Supported Standard score, Guaranteed messaging requirements score, Authentication Score, Authorization Score, Confidentiality Score, and Auditability Score and its value is the addition of these attribute values.

3.4.5 Enumeration QoS parameters

These parameters are user related factors such as user ratings, availability of QoS data to the user, and tooling options for the user. For instance, we can assign excellent, good, average, not bad and bad to these parameters (e.g. by using scales of 1-5).

3.4.5.1 *User Rating*

The user rating for a service is a measure of its usefulness for the user which is based on user's experiences of invoking different services. Users can have dissimilar ideas about one service. The value of user rating is defined as the average of different ratings by different users. We denote User Rating by UR (it will be measured by using scales of 1-5) [57].

3.4.5.2 *Availability of QoS Data to the User*

QoS data which is attained by the invocation of a service should be made available to the users in a way which guarantees that proper information is offered at abstraction levels with the intention of advancing the future planning and comprehension of service functionality [37]. This is a very important point to be considered in QoS evaluation of a composite service. We denote Availability of QoS Data to the User by AU (Availability will be measured by using scales of 1-5). This availability of QoS data to the user is different from the availability which is one of the probabilistic parameters.

3.4.5.3 *Tooling Options for the User*

A number of the key challenges are related to user-side tooling support in favor of the composite services' executions. The user should be able to simply state QoS constraints and link them to service compositions and definitions. Only users can evaluate those options [37]. Again user can assign a value (e.g. 1-5) to evaluate this parameter. We denote Tooling Options for the User by TU.

To calculate the final value of the discussed enumeration type QoS parameters for each web service composition, we simply employ the average value of their services' enumeration QoS scores.

Enumeration QOS Unique Value (EQUV) is defined as the combination of User Rating, Availability of QOS Data to the User, and the Tooling Options for the User and its value is the addition of these three attributes.

3.4.6 QoS Unique Value for the Composite Service

QoS Unique Value (QUV) is defined as the addition of Numeric Positive QoS Unique Value (NPQUV), Probabilistic QoS Unique Value (PQUV), Negative Numeric QoS Unique Value (NNQUV), Enumeration QoS Unique Value and Boolean Unique QoS Value (BUQV).

$$QUV = \mathbf{PQUV + BUQV + NPQUV + EQUV + NNQUV} \qquad \textbf{(Equation 37)}$$

This will give us a unique value for the composite QoS, which can have different range of values. We will normalize this value at the end to [0, 1]. Because NNQUV is defined as the addition of inverse values for latency, response time (L/R), and cost therefore it is an

increasing parameter. As a consequence based on what was explained, in case of having higher values, we will have a better QoS values for the composite service.

3.5 Combining QoS with Service Rank

Based on what was explained before, since we are looking for the optimal solution, we should search for the compositions with higher service rank unique scores. This means their nodes have a higher probability to be reached and invoked by the requester. This increases trust-ability because we cannot always rely on the QoS information. The probability of publishing false data by service providers (to promote their services) is the main reason for this distrust. Sometimes monitoring engines can be trusted but because of the high cost, some registries may not include the monitoring engines. On the other hand, we should have some defined requirements related to meet the QoS objectives demanded by the requester by using QoS-based calculations. Therefore we need to define a unique value to combine the two factors for ranking and selection of composite services.

To define Composition Service Unique Value (CSUV) we consider a coefficient α for Service Rank Unique Value (SUV) and a coefficient β for QUV, and the sum of them should be 1. By changing the coefficients, we can adjust our emphasis on each component. Later in the experiment, we will compare performances based on different values for them and compare the numeric results with simulated real time compositions QoS values. This comparison has to be accomplished in order to look for the best coefficients for ranking and selecting different compositions. Our Composition Service Unique Value is defined by the following formula:

$$\text{CSUV (WCS)} = \boldsymbol{\alpha} * \boldsymbol{SUV}(\boldsymbol{WCS}) + \boldsymbol{\beta} * \boldsymbol{QUV}(\boldsymbol{WCS}) \qquad \textbf{(Equation 38)}$$

3.6 Conclusion

In this chapter we have discussed our methodology, and how this method can be employed for web service composition and ranking to provide better ranking results returned to service designers. Another important goal of our defined algorithm is to facilitate the overall objectives of a web service composition process. The most important objective of every web service composition process is to find composition solutions which are the most reliable and trustable ones and satisfies user' QoS and functionality requirements in a manner that an atomic service is not capable of accomplishing that. We have discussed that through categorization and quantification of QoS values, greater assurance of QoS is able to be offered to the users earlier before deploying it into a real environment.

We defined two main analysis keys for ranking composite services. One approach is ranking composite web services based on QoS factors. The problem of QoS analysis is that some service providers aim to promote their services by publishing wrong QoS values. The other approach which could resolve this problem is employing some other algorithms independent of QoS factors such as social network analysis algorithm for ranking composite services. Our reasoning for this is to find the most reliable compositions at the end. We have developed a modified PageRank algorithm (which is called Service Rank in this work) in order to find the importance level of each service in the composition based on its connectivity and user invocation data; independent from QoS factors.

By employing invocation data, and users' contract data it is possible to assign weights to each link in the composition and employ our proposed Service Rank algorithm in order to calculate those weights. By considering all services' Service Rank scores we defined an SUV

for each composition solution and then combined it with our defined QUV, which is the combination of characterized and categorized QoS attributes, to calculate Composition Service Unique Value (CSUV). CSUV is calculated by the addition of SUV with coefficient α and QUV with a coefficient β. By adjusting these coefficients in the proposed formula for CSUV, we will be able to compare performances and the numeric outcomes with simulated real time composition QoS values in order to find the best option for ranking and selecting final composition solutions. This comparison has to be done in order to find the best coefficients to rank and select different compositions to effectively satisfy users' demands.

CHAPTER 4

EXPERIMENT

A key objective of the work in this thesis is to propose and implement a more reliable and trustable web service composition analysis and ranking approach. Our main idea is to rank different composition candidates based on not only their QoS values but also the previous usage history, so that the more frequently selected solutions could be ranked higher. In this chapter the proposed method for web services composition and ranking analysis is evaluated and the results are presented and discussed.

4.1 SR Tool

4.1.1 The Overview of Our Developed Analysis Tool

We designed and developed an analysis tool – SR tool (Service Rank tool) to offer a novel evaluation technique for web services composition, representation, and ranking analysis. This tool employs C++ as the main language to implement the proposed algorithms. SR tool has the ability of generating, analyzing, and visualizing web service composition candidates with different QoS values and web services registry graphs, and finally evaluating the performance of the selected web service compositions based on both QoS and social network analysis.

One of the principles in web service composition analysis is to guarantee the user satisfaction with the selected solutions. SR tool is conforming to this principle because it employs QoS data and social network analysis with the consideration of the usage history

data to guarantee the user satisfaction. SR tool is an interactive program which makes researchers able to try different web services registry graphs and QoS data to evaluate the composition ranking algorithms and obtain results in order to offer an optimized approach to find the most trustable and reliable web services compositions.

The SR tool first represents web services as nodes and their relationships as edges, and then the service composition solutions could be discovered from this large network of available registered services based on required input and output (adjacency matrices [50] [18], which their entries show the adjacency of two nodes, are employed to generate different graphs). The visualized representation of the registry networks can be generated using Graphviz [13] tool in each analysis. All the algorithms and methods explained in the previous chapter have been implemented in the SR tool. The new algorithms can be easily plugged into the tool for testing.

In the following sections, all the available functionalities, external files employed for the evaluation with this tool, its execution, and evaluation results will be described in details (external files will be explained in details in appendix A). Then the experiment results by using the SR tool will be shown in order to evaluate the effectiveness of our proposed Service Rank method. Also, the relations between the final rankings and different parameters in our experiments such as the coefficients (e.g. α and β in the proposed SUV and QUV combination formula) will be demonstrated.

4.1.2 Available Functionalities and External Files

Many of the algorithms explained before are implemented in our SR tool. A list of available functions is listed below:

1- *Main*: all of the calculations and table generations and interactions with users are defined here.

2- *ConvertToG*: employs the web services registry input adjacency matrix to support the visualization. This function creates a code which can be imported to Graphviz tool to visualize the registry graph in different formats.

3- *PageRank*: for calculating PageRank scores of web services.

4- *ServiceRankC*: calculates Service Rank scores of services based on the connectivity and weights calculated from the history graph.

5- *ConnectM*: creates a matrix which indicates which nodes are connected to a specific node.

6- *SizeofCTM*: calculates the maximum possible size of connections in the matrix.

7- *PTopBy*: calculates the number of nodes which point to a specific node, and also the number of nodes which points to that node.

8- *PathF*: finds all the possible routes between two nodes. The "PathF" function takes the input and output requirements from different users, and then searches for all possible composition solutions from the registry graph. In this searching process, the registry graph can be pruned in order to reduce the search space and the workload of searching exhaustively in the graph. The pruning is done based on input and output requirements to reduce the required time for analysis. Also, by defining the maximum number of solutions and length of solutions, which is an optional step (but highly recommended) it is possible to perform analysis in a shorter time. The function also sorts the solutions by their lengths. In the current stage, for the simplicity of discussion, we only consider the

sequential invocations (functional level composition's objective is finding a sequence of atomic services to satisfy a request [14]), other types of compositions can be extended later.

9- *Prune*: employed by "PathF" function in order to prune the search space.

10- *Linkage*: finds all linkages in the graph adjacency matrix and returns the size of it.

11- *EndNode*: finds if the last node of each travered path is the end node or not. This function is employed by the "PathF" function.

12- *RowSize*: returns the size of each row in the path matrix. This functions is used by the "PathF" function.

13- *RowReset*: resets the rows for the next path finding process. This functions is employed by the "PathF" function.

14- *SizePart*: returns the number of rows in a part of the path matrix. This function is used by the "PathF" function.

15- *FromFileToM*: reads a file and copies it to memory. This function is used to work with temporary files during the process of finding the routes in the registry graph.

16- *SizeFromF*: to allocate memory for the read file by the "FromFileToM" function.

17- *SortQ*: sorts the table of solutions based on calculated QUV values in "Main" function.

18- *SortQ2*: sorts the table of solutions based on a specific column of it.

19- *IntoDate*: converts minutes to dates in a string format, which is used for generating the history graph.

20- *Norm*: for normalizing values in a 2D matrix.

21- *Norm2*: for normalizing values in a column of the solution table.

There are many external files generated by this tool in each experiment saving all the necessary data for evaluation, analysis and visualization. Most of these files are in ".txt" format and they can be opened properly by WordPad program. The list and the explanations of external files are given in appendix A.

4.2 Experiment Design

4.2.1 Experiment Steps

SR tool is designed in a user friendly manner in order to offer the researchers the opportunity to consider many different parameters for comparison and evaluation for the purpose of dynamic, flexible and reliable web service composition and ranking. Our experiment is designed by using the SR tool in the following steps (Figure 10 and 11):

1- SR tool asks the user to enter the number of nodes in the registry graph.

2- SR tool asks the user to enter different parameters for calculating PageRank, e.g. the number of iterations, the damping factor, and the starting node number.

3- After getting required parameters to calculate PageRank scores, if the user requires a visualization script, it will generate the required script at the end. Visualization scripts assist to visualize the generated web service registry graph by using the Graphviz tool.

4- In this step user should enter the requested input and output of the required composition web service. Also, a maximum length and a maximum number of services for the composition can be entered to avoid generating too many solutions.

5- After getting the mentioned parameters, the program starts to generate the graph (if user chooses the automatic generation of the graph). The connectivity graph is totally random.

6- Based on the predefined value ranges for different QoS attributes, QoS values are randomly generated for each node. Then the tool creates external files to save the information about the graph matrix, visualization script, PageRank scores, routes between two nodes (candidate solutions), and the table with all services and their generated QoS values for further analysis. Up to this step, our SR tool can rank the composition candidates based on their QUV values or the combination of QUV and PUV values.

7- This step is to simulate the usage history which is provided from log. The SR tool will ask for the total number of previous invocations on those composition solutions, which means the same request has been served before. It will also ask for different percentage of usages on each solution. For the simplicity reason, we only consider the top three ranked composition solutions based on QUV values only and two other manually selected compositions which are usually ranked low in the original QUV-based ranking. The top three solutions are chosen because they are most likely to be viewed and selected based on some cognitive studies conducted on web search engines. The lower ranked solutions are chosen because we want to test whether our proposed Service Rank approach can promote those compositions if they are frequently invoked before. The other two required parameters are coefficients (α and β values) and the current date. After that it will calculate Service Rank scores and composition ranks based on

combined QUV and SUV from the generated usage history data and simulated usage percentages.

8- In this step it will generate the an external file ("simudat.txt") including important data such as original rankings based on QUV, and rankings based on combined PUV and QUV, and combined SUV and QUV.

9- In this step if user needs another simulation it will ask the user to enter new α and β values. All of the tried α, β values and the ranking results will be included in sumdat.txt file for evaluation and comparison purposes.

SR tool screen shots are illustrated in figures 10 and 11.

```
How many services do you have in your graph?50
Number of iterations for calculating Service Rank?50
Please enter damping factor:0.85
Please enter starting number:1
Do you need visualization script?y
please enter request start node (1 to 50):1

please enter request end node (1 to 50):50


Do you want to have a maximum length for solutions (Recommended)?n

Do you want to have a maximum number for solutions (Recommended)?y
please enter the maximum number of solutions:500

do you want your graph to be genrated (M)anually or (A)utomatically or from (F)i
le?a
```

Figure 10 – Screenshot 1 of SR tool

```
Do you need to have simulation?y
Please enter the number of users?1000

Please enter what percentage of users choose the first top three QUV solutions?5
0

Please enter what percentage of users choose the first QUV solution?20

Please enter what percentage of users choose the second QUV solution?20

The percentage of users choose the third QUV solution is 50.000000 - (20.000000
+ 20.0000000) which is 10.0000000
Please enter what solution will be selected in addition to the first three top s
olutions?20

Please enter what percentage of users choose the first solution in addition to t
op three?30

Please enter what second solution will be selected in addition to the first thre
e top solutions?21

The percentage of users choose the second solution in addition to top three is 2
0.000000
Please enter coefficient of SUV to calculate the final score:0.4

beta is 0.600000
Please enter current date year:2009

Please enter current date month:9

Please enter current date day:9


 SUV QUV simulation=1    solution#=228      Rank=1

 SUV QUV simulation=1    solution#=236      Rank=2

 SUV QUV simulation=1    solution#=81      Rank=3

 SUV QUV simulation=1    solution#=379      Rank=5

 SUV QUV simulation=1    solution#=373      Rank=17
do you want to try another values for alpha and beta?_
```

Figure 11 – Screenshot 2 of SR tool

4.2.2 Experiment Methodology

In each experiment top three compositions are selected based on QUV values and two others are selected manually with manually entered invocation percentages by the researcher. The usage data of these five compositions are included in the history usage file (historyl#.txt) with their QoS attribute values, component service nodes, users, invocation time, contract durations, and contract start dates.

In the beginning we have solutions sorted by their QUV values without considering usage data (soluqsorted.txt). Registry graph and its QoS attribute values in each experiment are generated randomly (it is also possible to be entered manually or from a file). Therefore, it is possible to perform each experiment for different graph structures and QoS attribute values for the evaluation purpose.

After entering coefficients (α and β) values in the final formula (for each step), a history usage data will be generated based on the usage number and percentage of usages entered in the execution time. Generated usage data can be employed to calculate weights using the formulas explained in the previous chapter. These weights will be employed to calculate Service Rank score for each web service.

SUV value for each composition is calculated based on its component nodes' Service Rank scores. Final rankings are based on the combination of SUV and QUV, and the average ranking of the five chosen compositions will be compared with the QUV only based rankings in order to evaluate the ranking improvements. If a lower average ranking is achieved than QUV only approach, then the results demonstrate the improvement in ranking compositions. Usually different α and β values produce different results, and by comparing results from different α and β combinations, we could also find a set of coefficients offering the best optimized rankings.

Combined PUV and QUV scores also will be considered to see the effect of graph structure in the final rankings. The reason is that the combination of PUV and QUV is independent from the history data and it is based on both web services registry graph's connectivity and its QoS attribute values. This consideration gives a better understanding to

evaluate the effect of SUV scores on final rankings because SUV is based on both graph structure and the usage data. Having a higher PUV demonstrates a stronger connectivity of each composition node. Weak connectivity of composition nodes will decrease the improvements caused by combined SUV and QUV.

As explained before, “simudat.txt” file contains all the necessary data for the final evaluation. In continue some of the experiment results will be explained to demonstrate the effectiveness of the proposed algorithm in this thesis for web service composition and ranking.

4.3 Experiments and Result Analysis

Four experiments are demonstrated here. They represent different simulation scenarios. They have different web service registry graphs, different numbers of invocations and different QoS attribute values. Experiments 1, 2 and 3 have different service graphs. Experiment 4 has the same web service graph as experiment 3 in order to study the effect of increasing invocation times on our ranking method. Therefore the number of usages in experiment 3 is set to 100, and in the other three experiments is 1000. QoS attribute values in all four experiments are different. These data sets assist to evaluate the proposed algorithm for the same registry graphs with different QoS values (e.g. by comparing results from experiments 3 and 4). Therefore, by considering original rankings based on QUV only, and considering the combination of SUV and QUV rankings, it is possible to demonstrate the effect of combining SUV in ranking web service compositions with QUV.

4.3.1 Experiments

Experiment #1:

In this experiment we have 30 nodes in the web services registry graph. Figure 13 illustrates the structure of this graph using the Graphviz tool (Figure 12). This way of visualization assists to have a better perspective of each web service registry graph and to observe web services' inputs and outputs. This illustration is based on the script generated by the SR tool (Figure 12).

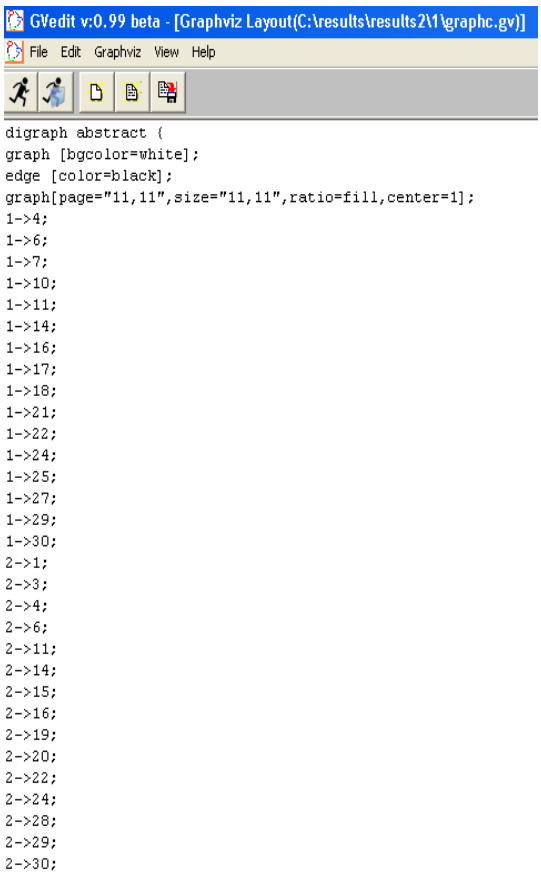


Figure 12– Graphviz tool screenshot with code generated by SR tool

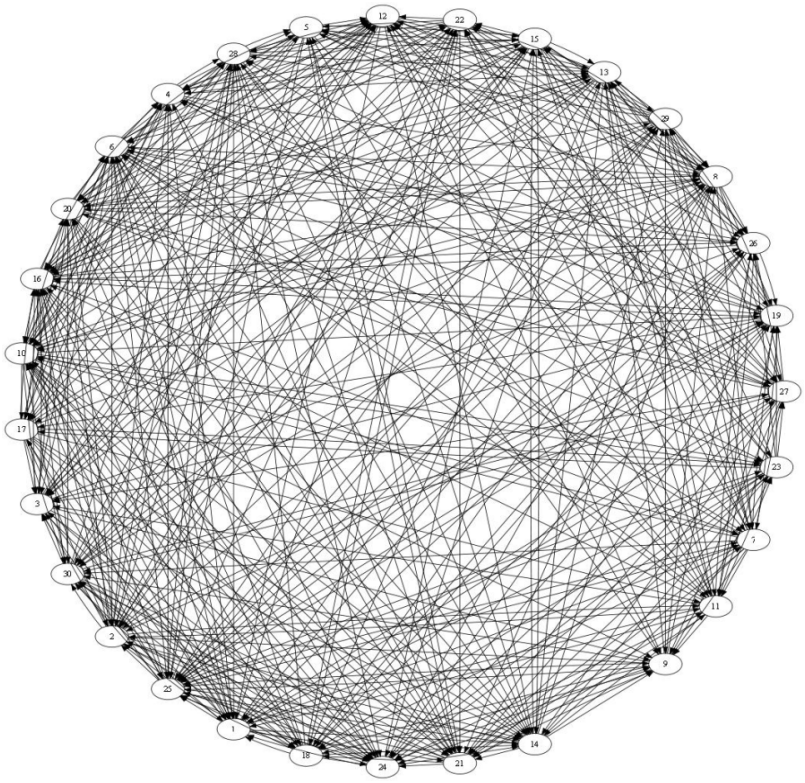


Figure 13 – Graph visualization for experiment #1 by Graphviz tool

More figures to illustrate the experiment are included in Appendix B (B.1-B.9). These figures show PageRank scores, Service Rank scores of all services, the service QoS values, the history data, and the candidate composition solutions sorted by QUV value only, QUV plus PUV, and QUV plus SUV with different coefficient values. Final results and rankings will be stored in "simudat.txt" file (Figure 14). We have 1000 invocations for the simulation demonstrated in Figure 14.

```
 Percentage of users which choose the 20 th ranked composition based on QUV (solution number 147)is: 35.000000
 Percentage of users which choose the 21 th ranked composition based on QUV (solution number 175) is: 25.000000

                                        *****simulation=3*****
alpha=0.200000 beta=0.800000
rankings based on QUV:
 solution# 174 --> rank: 1
 solution# 129 --> rank: 2
 solution# 18 --> rank:3
 solution# 147 --> rank: 20
 solution# 175 --> rank:21
 average ranking is: 9.400000
alpha*PUV and beta*QUV rankings:
 solution# 174 --> rank: 1
 solution# 129 --> rank:2
 solution# 18 --> rank: 5
 solution# 147 --> rank: 25
 solution# 175 --> rank: 26
average of used solutions rankings based on alpha*PUV and beta*QUV: 11.800000
alpha*SUV and beta*QUV rankings:
 solution# 174 --> rank: 2
 solution# 18 --> rank:3
 solution# 129 --> rank: 4
 solution# 147 --> rank: 11
 solution# 175 --> rank: 12
average of used solutions rankings based on alpha*SUV and beta*QUV: 6.400000
number of users is: 1000
 percentage of users which choose the first ranked composition based on QUV (solution number 174)is: 18.000000
 Percentage of users which choose the second ranked composition based on QUV (solution number 129)is: 12.000000
 Percentage of users which choose the third ranked composition based on QUV (solution number 18)is: 10.000000
 Percentage of users which choose the 20 th ranked composition based on QUV (solution number 147)is: 35.000000
 Percentage of users which choose the 21 th ranked composition based on QUV (solution number 175) is: 25.000000

                                        *****simulation=4*****
alpha=0.300000 beta=0.700000
rankings based on QUV:
 solution# 174 --> rank: 1
 solution# 129 --> rank: 2
 solution# 18 --> rank:3
```

Figure 14 - Part of the simulation results with rankings for experiment #1 generated by SR tool

Experiment #2:

In this experiment we have 40 nodes in the web services registry graph. Figure B.10 illustrates the structure of this graph using the Graphviz tool. All other information such as web services' data is illustrated in Figures B.11-B.19. There are 1000 times of invocations in this simulation.

Experiment #3:

In this experiment we have 30 nodes in the web services registry graph. Figure B.20 illustrates the structure of this graph using the Graphviz tool. Figure 15 shows the final simulation data for this experiment. There are 100 times of invocations in this simulation as can be observed in Figure 15.

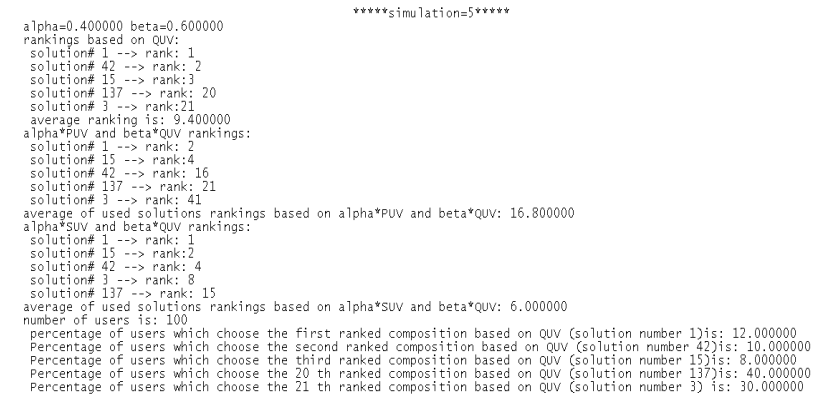


Figure 15 - Part of simulation results for experiment #3 generated by SR tool

Experiment #4:

In this experiment web services registry graph is same as experiment #3. Figure 16 demonstrates the final simulation data for this experiment. There are 1000 times of invocations in this simulation as can be observed in Figure 16. In the following section, the results of different experiments are visualized, analyzed and compared for the final evaluation and conclusion.

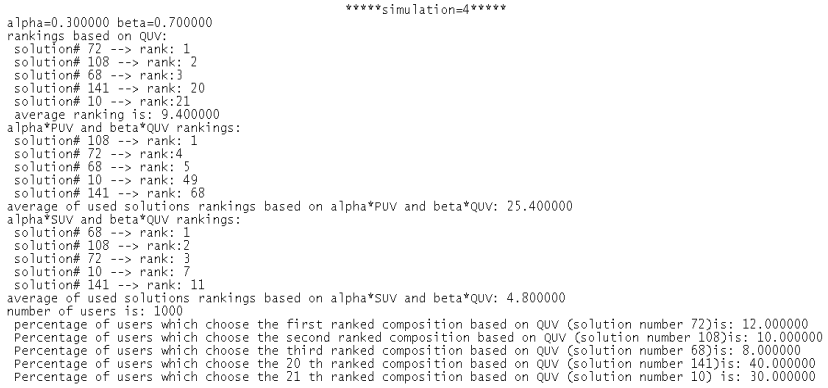


Figure 16 - Part of simulation results for experiment #4 generated by SR tool

4.3.2 Comparison and Evaluation of the Results

As mentioned before, in each experiment different α and β values are entered to employ the proposed ranking algorithm to calculate final score for each composition and therefore get a new ranking for the original top 30 results based on QUV value only. The average ranks of the top three compositions based on QUV only and also two other manually selected compositions are calculated. In addition to the combination of SUV and QUV scores, combination of PUV and QUV values are considered in order to have a better understanding about the effect of SUV on new rankings.

Based on the experiment results (Tables 3 to 14), combination of SUV and QUV improves the ranking for the previously selected and invoked compositions (which will be calculated and evaluated). In each scenario, certain α and β values give an optimized ranking result. Also, by comparing experiments #3 and #4 results (they have same web services registry graphs), it can be demonstrated that by increasing the number of invocations (e.g. more history data available) the ranking can be improved even more. Results from 4 experiments are organized and presented in the following tables.

Table 3 shows our simulated usage percentage data for top three and two other manually chosen solutions (i.e. 20 and 21). Left column shows the solution numbers, middle column shows the original ranks based on QUV data, and the right column shows the usage percentages (e.g. 10 means 10% users chose this solution).

Table 3 - Usage data for experiment #1

Solution #	Original Rank Based on QUV	% of Usage
18	3	10
129	2	12
147	20	35
174	1	18
175	21	25

Table 4 shows rankings based on combination of PUV and QUV scores (PQ#). We could see that when the coefficient on PUV is higher, the average rank for those 5 solutions is much higher. It infers that the ranking based on the connectivity of the services is completely different from the ranking based on QoS values. So simply combining the two together is not very helpful.

Table 5 illustrates the rankings based on SUV and QUV values (SQ#). It is possible to evaluate the improvements by comparing the average rankings with different values for coefficients. When $\alpha=0$, the ranking is based on QUV only. We could see that the average rank for those 5 solutions is 9.4 in the original QUV-based ranking. When we combine QUV with SUV, the average rank is consistently lower than 9.4. The best performance can be achieved when $\alpha=0.2$ and $\beta=0.8$ with the average rank 6.4. The percentage of improvement in average ranking of selected compositions (for the best combination of SUV and QUV) is:

$\frac{9.4-6.4}{9.4}\approx 0.32$ → therefore the highest ranking improvement is 32% for experiment #1.

Table 4 - Rankings based on combined normalized PUV and QUV for experiment #1

Solution Number	QUV Based Rank	PQ# α=0, β=1	PQ# 0.1, 0.9	PQ# 0.2, 0.8	PQ# 0.3, 0.7	PQ# 0.4, 0.6	PQ# 0.5, 0.5	PQ# 0.6, 0.4	PQ# 0.7, 0.3	PQ# 0.8, 0.2	PQ# 0.9, 0.1	PQ# 1, 0
18	3	3	3	5	7	14	38	84	140	156	174	182
129	2	2	2	2	3	6	13	45	93	126	146	159
147	20	20	23	25	30	53	98	148	165	179	185	186
174	1	1	1	1	3	4	9	41	94	130	149	164
175	21	21	22	26	11	52	97	149	166	178	184	187
Average	9.4	9.4	10.2	11.8	14.4	25.8	51	93.4	131.6	153.8	167.6	175.6

Table 5 - Rankings based on combined normalized SUV and QUV for experiment # 1

Solution Number	QUV Based Rank	SQ# α=0, β=1	SQ# 0.1, 0.9	SQ# 0.2, 0.8	SQ# 0.3, 0.7	SQ# 0.4, 0.6	SQ# 0.5, 0.5	SQ# 0.6, 0.4	SQ# 0.7, 0.3	SQ# 0.8, 0.2	SQ# 0.9, 0.1	SQ# 1, 0
18	3	3	4	3	5	5	5	5	6	6	6	6
129	2	2	3	4	6	7	6	6	7	7	8	8
147	20	20	16	11	10	10	10	10	10	9	9	9
174	1	1	1	2	3	6	7	8	9	11	11	13
175	21	21	17	12	11	11	11	11	11	10	10	10
Average	9.4	9.4	8.2	6.4	7	7.8	7.8	8	8.6	8.6	8.8	9.2

Table 6 shows the solution numbers, original rankings based on QUV data, and usage percentages of selected compositions for experiment #2.

Table 6 - Usage data for experiment #2

Solution #	Original Rank Based on QUV	% of Usage
36	20	34
56	1	20
57	3	10
61	2	20
194	21	16

Table 7 and Table 8 show the result based on combined PUV and QUV and combined SUV and QUV respectively. From Table 7, it is clear that by increasing the coefficient of PUV, rankings of selected solutions becomes worse. This result shows that PUV rankings can be totally different from QUV rankings and they are just based on the connectivity of the graph. On the other hand, SUV rankings are based on both connectivity and usage history data, which we believe is more accurate.

By looking at results in Table 8, we could conclude that combination of SUV and QUV again gives us some improvement in the final rankings. The percentage of improvement for experiment #2 is (for the best case α=0.4 and β=0.6):

$\frac{9.4-4.4}{9.4} \approx 0.53$ → therefore the highest ranking improvement is 53% for experiment #2.

Table 7 - Rankings based on combined normalized PUV and QUV for experiment #2

Solution Number	QUV Based Rank	PQ# α=0, β=1	PQ# 0.1, 0.9	PQ# 0.2, 0.8	PQ# 0.3, 0.7	PQ# 0.4, 0.6	PQ# 0.5, 0.5	PQ# 0.6, 0.4	PQ# 0.7, 0.3	PQ# 0.8, 0.2	PQ# 0.9, 0.1	PQ# 1, 0
36	20	20	17	17	17	19	20	26	33	43	57	62
56	1	1	2	3	6	12	32	69	113	142	166	184
57	3	3	3	2	1	2	4	6	13	21	29	41
61	2	2	1	1	2	5	7	17	43	72	96	113
194	21	21	19	18	18	20	23	30	39	52	61	67
Average	9.4	9.4	8.4	8.2	8.8	11.6	17.2	29.6	48.2	66	81.8	93.4

Table 8 - Rankings based on combined normalized SUV and QUV for experiment # 2

Solution Number	QUV Based Rank	SQ# α=0, β=1	SQ# 0.1, 0.9	SQ# 0.2, 0.8	SQ# 0.3, 0.7	SQ# 0.4, 0.6	SQ# 0.5, 0.5	SQ# 0.6, 0.4	SQ# 0.7, 0.3	SQ# 0.8, 0.2	SQ# 0.9, 0.1	SQ# 1, 0
36	20	20	16	8	7	6	6	6	5	5	5	5
56	1	1	2	2	2	2	4	4	4	4	4	4
57	3	3	3	4	5	5	5	5	6	6	7	9
61	2	2	1	1	1	1	1	1	2	3	3	3
194	21	21	18	15	9	8	7	8	9	10	10	11
Average	9.4	9.4	8	6	4.8	4.4	4.6	4.8	5.2	5.6	5.8	6.4

Table 9 shows the usage percentages for experiment #3. Table 10 shows the rankings based on combined PUV and QUV and Table 11 shows rankings based on combined SUV

and QUV. This experiment will be used to compare with experiment #4 which has different number of invocations and same settings of all other parameters.

Table 9 - Usage data for experiment #3

Solution #	Original Rank Based on QUV	% of Usage
1	1	12
3	21	30
15	3	8
42	2	10
137	20	40

Table 10 - Rankings based on combined normalized PUV and QUV for experiment #3

Solution Number	QUV Based Rank	PQ# α=0, β=1	PQ# 0.1, 0.9	PQ# 0.2, 0.8	PQ# 0.3, 0.7	PQ# 0.4, 0.6	PQ# 0.5, 0.5	PQ# 0.6, 0.4	PQ# 0.7, 0.3	PQ# 0.8, 0.2	PQ# 0.9, 0.1	PQ# 1, 0
1	1	1	1	1	1	2	7	16	35	66	89	112
3	21	21	23	27	32	41	56	74	94	119	138	148
15	3	3	3	2	3	4	8	10	26	43	61	82
42	2	2	2	6	9	16	35	59	89	129	150	163
137	20	20	18	17	20	21	30	40	49	72	85	96
Average	9.4	9.4	9.4	10.6	13	16.8	27.2	39.8	58.6	85.8	104.6	120.2

As mentioned before we have 100 invocations for this experiment and as it can be observed from Table 11 the improvement in ranking for experiment #3 (for the best combination of SUV and QUV with α=0.5 and β=0.5) is:

$\frac{9.4-5.6}{9.4} \approx 0.40$ → therefore the highest ranking improvement is 40% for selected compositions in experiment #3.

Table 11 - Rankings based on combined normalized SUV and QUV for experiment #3

Solution Number	QUV Based Rank	SQ# α=0, β=1	SQ# 0.1, 0.9	SQ# 0.2, 0.8	SQ# 0.3, 0.7	SQ# 0.4, 0.6	SQ# 0.5, 0.5	SQ# 0.6, 0.4	SQ# 0.7, 0.3	SQ# 0.8, 0.2	SQ# 0.9, 0.1	SQ# 1, 0
1	1	1	1	1	1	1	1	1	1	1	1	1
3	21	21	16	9	8	8	7	8	8	10	10	9
15	3	3	2	2	2	2	2	2	2	2	2	2
42	2	2	3	3	4	4	4	4	6	7	9	12
137	20	20	18	15	15	15	14	16	17	18	21	26
Average	9.4	9.4	8	6	6	6	5.6	6.2	6.8	7.6	8.6	10

Table 12 shows usage data for experiment #4. The percentage of usages is the same as experiment #3 for selected compositions. We have different solution numbers because we have different QUV values for different compositions in these two experiments, and it doesn't affect our analysis.

Table 12 - Usage data for experiment #4

Solution #	Original Rank Based on QUV	% of Usage
10	21	30
68	3	8
72	1	12
108	2	10
141	20	40

Table 13 demonstrates the rankings based on combined PUV and QUV and Table 14 shows the rankings based on combined SUV and QUV for experiment #4. By comparing Table 14 with Table 11 we are able to observe the effect of increasing invocation numbers on ranking improvements.

As mentioned before we have 1000 invocations for experiment #4 and based on Table 14 the percentage of improvement is (for α=0.7 and β=0.3):

$\frac{9.4-4.2}{9.4} \approx 0.55$ → the highest ranking improvement is 55% for experiment #4 (better than experiment #3 when the number of invocations is 100)

Table 13 - Rankings based on combined normalized PUV and QUV for experiment #4

Solution Number	QUV Based Rank	PQ# α=0, β=1	PQ# 0.1, 0.9	PQ# 0.2, 0.8	PQ# 0.3, 0.7	PQ# 0.4, 0.6	PQ# 0.5, 0.5	PQ# 0.6, 0.4	PQ# 0.7, 0.3	PQ# 0.8, 0.2	PQ# 0.9, 0.1	PQ# 1, 0
10	21	21	25	35	49	66	89	105	128	148	156	164
68	3	3	3	5	1	5	11	27	55	86	112	139
72	1	1	2	2	4	8	17	47	88	131	157	167
108	2	2	1	1	1	1	1	1	1	3	7	13
141	20	20	27	46	68	88	113	141	167	170	179	183
Average	9.4	9.4	11.6	17.8	25.4	33.6	46.2	64.2	87.8	107.6	122.2	133.2

Table 14 - Rankings based on combined normalized SUV and QUV for experiment #4

Solution Number	QUV Based Rank	SQ# α=0, β=1	SQ# 0.1, 0.9	SQ# 0.2, 0.8	SQ# 0.3, 0.7	SQ# 0.4, 0.6	SQ# 0.5, 0.5	SQ# 0.6, 0.4	SQ# 0.7, 0.3	SQ# 0.8, 0.2	SQ# 0.9, 0.1	SQ# 1, 0
10	21	21	11	10	7	7	7	7	5	4	2	2
68	3	3	3	1	1	1	1	1	1	1	1	1
72	1	1	2	3	3	3	3	4	4	6	8	8
108	2	2	1	2	2	2	2	2	2	3	4	5
141	20	20	14	12	11	9	8	9	9	9	9	9
Average	9.4	9.4	6.2	5.6	4.8	4.4	4.2	4.6	4.2	4.6	4.8	5

Table 15 shows the comparison between experiment #3 and #4, which have same graphs but different number of invocations. SUV value for experiment #3 is 10 whereas in experiment #4 it is 5, and it has been improved by 50% when the number of invocations is increased from 100 to 1000. And the best SUV+QUV combination is also improved from 5.6 to 4.2, which is 25% increase. It can be concluded that the ranking result is improved when there are more invocations in the history file. This feature demonstrates the effectiveness of applying weights in the proposed Service Rank algorithm.

Table 15 - A sample comparison to demonstrate the effect of increasing invocations on final rankings

Experiment #	Invocation #	SUV	Best SQ#
3	100	10	5.6
4	1000	5	4.2

Our experiment also shows the PUV value could be totally different from QUV based ranking. Therefore it is not enough to consider only PageRank values or QoS values as in many QoS papers, and it is useful to consider usage data with PageRank. Combining SUV with QUV can improve ranking performance significantly. Also, one set of α and β values can offer the optimal results.

4.4 Conclusion

SR tool is an analysis tool based on proposed algorithms in this thesis for the purpose of web service composition and ranking. It assists to find the importance level of each service (node) in the composition based on its connectivity and history data. History log file

generated by our SR tool contains user invocation data and contract data which make it possible to assign higher weights to links with higher number of invocations, later invocation time, contract dates, and longer contract durations. SR tool calculates SUV score of each composition and combines it with QUV score to calculate the Composition Service Unique Value (CSUV).

CSUV is calculated by adding SUV with a coefficient α and QUV with a coefficient β. These coefficients in the proposed formula for CSUV can be adjusted to compare performances and the numeric outcomes with simulated real time composition QoS values in order to find the best coefficients to rank and select different compositions to effectively satisfy users' demands in a trustable and reliable manner.

Comparing four experiment results in this chapter demonstrates the effectiveness of the Service Rank algorithm for ranking compositions. These results also show that by increasing the number of invocations, performance of the Service Rank algorithm could be further improved.

CHAPTER 5

CONCLUSIONS

5.1 Conclusions

The necessity of discovering an approach for reliable and trustable automatic web service composition and ranking has been addressed by many researchers. Until now, several attempts have been made in this field for designing techniques and supporting tools to achieve required objectives such as selecting reliable compositions with guaranteed QoS levels. However, offering reliable models of web service composition, ranking, verification, and evaluation with considering possible false QoS attribute values have been mostly ignored in pervious works.

The major objective of this thesis is to offer an algorithm, which if implemented in a tool, demonstrates an effective way for the purpose of automatic web services composition and ranking in a reliable and trustable manner. QoS attribute values alone are not completely trustable since some providers publish wrong QoS attribute values in order to promote their services. The graph representing the relationship among web services forms the basis of employing social analysis techniques for service composition and ranking analysis. The usage history data could further improve the accuracy of social analysis techniques. Both composition ranking mechanisms (e.g. QoS-based and usage-biased social network analysis) are implemented in this thesis in a combined manner in order to achieve the optimal performance.

Our contributions include a number of particular features. Firstly, graph modeling of web services registry and composition with implemented procedures provide a way to analyze them directly without considering QoS factors. This feature improves the trust-ability of selected compositions in case of having manipulated QoS attribute values. Secondly, a combined algorithm is offered to rank the final compositions based on both QoS parameters and Service Rank scores. Service Rank scores are based on both the connectivity of web services (which is based on interaction behavior among services) and usage history data. This characteristic makes proposed algorithm suitable to be employed for giving a higher rank to more reliable compositions which have been selected and invoked before. Thirdly, required history data parameters are defined and quantified for being capable of providing a higher rank to more popular and up to date compositions. This objective can be achieved by calculating weights of links based on the history data and employing the weights to calculate Service Rank scores for each composition. Fourthly, QoS attribute values are categorized and quantified to consider all of the possible QoS factors in the final QoS analysis. This categorization and quantification demonstrate the flexibility and comprehensiveness of the proposed approach of calculating the QoS-based rank scores. Finally, an analysis tool has been designed and developed for web service composition and ranking based on proposed algorithms which we hope could help research communities to find the best web service composition and ranking approach that can be employed for real cases.

5.2 Future Works

There are many works which can be done along this research direction. Firstly, we would like to extend our experiment to other invocation types such as concurrent or selection or

join. And we would also want to have simulated data closer to the real scenario (e.g. usage of different compositions from more diversified users). Secondly, we could explore other techniques for web service composition and ranking such as AI techniques to see whether and how it could be integrated with our approach to further improve the performance. Thirdly, different data representations for the selected compositions can be implemented in order to be employed with other techniques. Lastly, the approach proposed in this work can be tested and evaluated in real applications with real web service registry graphs and QoS attribute values for future works.

REFERENCES

1. Agosti, M.; Pretto, L., “A Theoretical Study of a Generalized Version of Kleinberg's HITS Algorithm,” Information Retrieval 8, 2 (Apr. 2005), pp. 219-243.
2. Ai, L.; Tang, M., “QoS-Based Web Service Composition Accommodating Inter-service Dependencies Using Minimal-Conflict Hill-Climbing Repair Genetic Algorithm,” In Proceedings of the 2008 Fourth IEEE international Conference on Escience (December 07-12, 2008). ESCIENCE. IEEE Computer Society, Washington, DC, pp. 119-126.
3. Akkus, G.B., "Semantic Web Services Composition: A Network Analysis Approach," Data Engineering Workshop, 2007 IEEE 23rd International Conference on, pp.937-943, 17-20 April 2007
4. Aydogan, R.; Zirtiloglu, H., "A Graph-BasedWeb Service Composition Technique Using Ontological Information," Web Services, 2007. ICWS 2007. IEEE International Conference on, pp.1154-1155, 9-13 July 2007
5. Berbner, R.; Spahn, M.; Repp, N.; Heckmann, O.; Steinmetz, R., "Heuristics for QoS-aware Web Service Composition," Web Services, 2006. ICWS '06. International Conference on, pp.72-82, 18-22 Sept. 2006
6. Cardellini, V.; Casalicchio, E.; Grassi, V.; Lo Presti, F., "Flow-Based Service Selection for Web Service Composition Supporting Multiple QoS Classes," Web Services, 2007. ICWS 2007. IEEE International Conference on, pp.743-750, 9-13 July 2007

7. Chang, H.C.; Wang, J.; Chiu, C.Y.; "Automatic Semantic Web Service Composition via Agent Intention Execution in AgentSpeak," Web Intelligence, IEEE/WIC/ACM International Conference on, no., pp.564-567, 2-5 Nov. 2007
8. Colbourn, C.J.; Chen, Y.; Tsai, W.T., "Progressive ranking and composition of Web services using covering arrays," Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on , vol., no., pp. 179-185, 2-4 Feb. 2005
9. D'Ambrogio, A., "A Model-driven WSDL Extension for Describing the QoS ofWeb Services," Web Services, 2006. ICWS '06. International Conference on, pp.789-796, 18-22 Sept. 2006
10. De Paoli, F.; Lulli, G.; Maurino, A., "Design o f Quality-based Composite Services," In Proceedings o f 4th International Conference on Service Oriented Computing - ICSO C 2006 - Chicago, USA LNCS vol. 4294, pp. 153- 164, 2006.
11. Domingos, P.; Richardson, M., "Mining the network value of customers ,"In Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 57- 66, San Francisco , USA, August 2001.
12. Farahat, A.; Lofaro, T.; Miller, J.C., Rae, G.; Ward, L.A., "Authority rankings from hits, pagerank, and salsa: Existence, uniqueness, and effect of initialization," SIAM J. Sci. Comput., vol. 27, no. 4, pp. 1181-1201, 2006.
13. Fox, C.; Wilson, D., "Visualization in Interrogator using Graphviz," Information Assurance Workshop, 2006 IEEE , vol., no., pp.390-392, 21-23 June 2006

14. Freddy, L.; Alain, L., "Semantic Web Service Composition Based on a Closed World Assumption," Web Services, 2006. ECOWS '06. 4th European Conference on, pp.233-242, Dec. 2006

15. Gekas, J.; Fasli, M., “Employing Graph Network Analysis for Web Service Composition,” In Alkhatib G. I. and Rine, D. C. (eds), Agent Technologies and Web Engineering, IGI Global Publishers, December 2008 International Journal of Information

    Technology and Web Engineering, Vol. 2. Issue. 4. pp. 21 - 40

16. Gu, Z.; Li, J.; Xu, B., "Automatic Service Composition Based on Enhanced Service Dependency Graph," Web Services, 2008. ICWS '08. IEEE International Conference on, pp.246-253, 23-26 Sept. 2008

17. Guo, L.Y.; Chen, H.P.; Yang, G.; Fei, R.Y., "A QoS Evaluation Algorithm for Web Service Ranking Based on Artificial Neural Network," Computer Science and Software Engineering, 2008 International Conference on, pp.381-384, 12-14 Dec. 2008

18. Hanneman, R.A.; Riddle, M., “Introduction to social network methods,” Riverside, CA: University of California, Riverside, 2001. Published in digital form at http://faculty.ucr.edu/~hanneman/).

19. Hashemian, S.V.; Mavaddat, F., "A graph-based approach to Web services composition," Applications and the Internet, 2005. Proceedings. The 2005 Symposium on, pp. 183-189, 31 Jan.-4 Feb. 2005

20. Hovakimyan, A.; Sargsyan, S.; Barkhoudaryan, S., "Genetic algorithm and the problem of getting knowledge in e-learning systems," Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference on, pp. 336-339, 30 Aug.-1 Sept. 2004
21. Jensen, D., “Statistical challenges to inductive inference in linked data,” Preliminary papers of the 7th International Workshop on Artificial Intelligence and Statistics; 1999 Jan 4 - 6; Fort Lauderdale. FL.
22. Kalasapur, S.; Kumar, M.; Shirazi, B.A., "Dynamic Service Composition in Pervasive Computing," Parallel and Distributed Systems, IEEE Transactions on , vol.18, no.7, pp.907-918, July 2007
23. Kalepu, S.; Krishnaswamy, S.; Loke, S.W., "Verity: a QoS metric for selecting Web services and providers," Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on, pp. 131-139, 13 Dec. 2003
24. Kempe, D.; Kleinberg, J.; Tardos, E., “Maximizing the Spread of Influence through a Social Network,” Proc. 9th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining, ACM Press, 2003, pp. 137–146.
25. Kim, G.; Christos, F.; Martial, H., "Unsupervised modeling of object categories using link analysis techniques," Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pp.1-8, 23-28 June 2008
26. Kleinberg, J., “Authoritative sources in a hyperlinked environment,” Journal of the ACM, 1999, 46, 5:604-632.

27. Lei, X.; Jing, L.; Jie, Q.; Pershing, J.A.; Ying, L.; Ying C., "Availability “weak point” analysis over an SOA deployment framework," Network Operations and Management Symposium, 2008. NOMS 2008. IEEE, pp.473-480, 7-11 April 2008
28. Li, L.; Ma, J.; Chen, Z.M.; Ling, S., "An Efficient Algorithm for Web Services Composition with a Chain Data Structure," Services Computing, 2006. APSCC '06. IEEE Asia-Pacific Conference on, pp.64-69, Dec. 2006
29. Liang Q.; Yang, S., "AND/OR Graph and Search Algorithm for Discovering Composite Web Services, " International Journal of Web Services Research, Vol. 2. No. 4. pp. 48 -67, 2005
30. Liu, J.; Lian, C., "Web Services as a Graph and Its Application for Service Discovery," Grid and Cooperative Computing, 2006. GCC 2006. Fifth International Conference on, no., pp.293-300, Oct. 2006
31. Lu, H.E., "Ranking Web services based on ontology semantics," Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on, pp.2161-2165 Vol. 4, 18-21 Aug. 2005
32. Maimon, O.; Rokach, L., “Data Mining and Knowledge Discovery Handbook,” Springer-Verlag New York, Inc, pp. 417-432.
33. Mei, L.; Zhang, Z.; Chan, W.K.; T.H., T., "An Adaptive Service Selection Approach to Service Composition," Web Services, 2008. ICWS '08. IEEE International Conference on, pp.70-77, 23-26 Sept. 2008
34. Menasce, D.A., “Composing Web Services: A QoS View,” IEEE Internet Computing, v.8 n.6, p.88-90, November 2004

35. Menascé, D.A., "QoS Issues in Web Services," IEEE Internet Computing, vol. 6, no. 6, Nov./Dec. 2002, pp. 72-75.
36. Montgomery, D.C.; Runger, G.C., “Applied Statistics and Probability for Engineers, 3rd Edition, and JustAsk!,” Set (John Wiley & Sons, 2006), third edition, pp. 42-52
37. Mos, A., “Challenges in Integrating Tooling and Monitoring for QoS Provisioning in SOA Systems (Keynote). In Service-Oriented Computing,” (2009), ICSOC 2008 Workshops: ICSOC 2008 international Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers, G. Feuerlicht and W. Lamersdorf, Eds. Lecture Notes In Computer Science, vol. 5472. Springer-Verlag, Berlin, Heidelberg, 189-189.
38. Mukherjee, D.; Jalote, P.; Gowri Nanda, M., "Determining QoS of WS-BPEL compositions," 6th International Conference on Service-Oriented Computing - ICSOC 2008, volume 5364 of LNCS, pp. 378–393, 2008.
39. Mustafa, F.; McCluskey, T.L., "Dynamic Web Service Composition," Computer Engineering and Technology, 2009. ICCET'08. International Conference on , vol.2, pp.463-467, 22-24 Jan. 2009
40. Narayanan, S.; McIlraith, S., “Simulation, Verification, and Automated Composition of Web Services,” Proc. 11th Int’l World Wide Web Conf. (WWW 02), ACM Press, 2002, pp. 77–88.
41. Pretto, L., “A theoretical analysis of Google’s PageRank,” In Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE), volume 2476 of Lecture Notes in Computer Science (LNCS), pp. 131–144. Springer-Verlag, 2002.

42. Ran, S., "A model for web services discovery with QoS," SIGecom Exch., vol. 4, no. 1, pp. 1-10, 2003.
43. Rao, J.; X. Su., “A Survey of Automated Web Service Composition Methods”, Springer, 2004, p. 43-54.
44. Richardson, M.; Domingos, P., “Mining Knowledge-Sharing Sites for Viral Marketing,” Proc. 8th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining, ACM Press, 2002, pp. 61–70.
45. Shiaa, M.M.; Fladmark, J.O.; Thiell, B., "An Incremental Graph-based Approach to Automatic Service Composition," Services Computing, 2008. SCC '08. IEEE International Conference on, pp.397-404, 7-11 July 2008
46. Singhera, Z.U., "Extended Web services framework to meet non-functional requirements," Applications and the Internet Workshops, 2004. SAINT 2004 Workshops. 2004 International Symposium on, pp. 334-340, 26-30 Jan. 2004
47. Thissen, D.; Wesnarat, P., "Considering QoS Aspects in Web Service Composition," Computers and Communications, 2006. ISCC '06. Proceedings. 11th IEEE Symposium, pp. 371-377, 26-29 June 2006.
48. Toma, I.; Roman, D.; Fensel, D., "On Describing and Ranking Services based on Non-Functional Properties," Next Generation Web Services Practices, 2007. NWeSP 2007. Third International Conference on, pp.61-66, 29-31 Oct. 2007.
49. Vladimir, S.; Christian, S., “Negotiating and enforcing QoS and SLAs in grid and cloud computing,” In GPC ’09: Proceedings of the 4th International Conference on

Advances in Grid and Pervasive Computing, pp. 25–35, Berlin, Heidelberg, 2009. Springer-Verlag.

50. Wasserman, S.; Faust, K., “Social Network Analysis,” Cambridge University Press 1994, pp. 150-152.
51. Wu, Q.; Arun, I.; Subramanian, R.; Rouvellou, I.; Silva-Lepe, I.; Mikalsen, T., “Combining Quality of Service and Social Information for Ranking Services,” In Proceedings of ICSOC-ServiceWave 2009 (ICSOC'09), pp. 561-575, November 24-27 2009, Stockholm, Sweden.
52. Xia, J., "QoS-Based Service Composition," compsac, vol. 2, pp.359-361, 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006
53. Xu, J. J.; Chen, H., “Using shortest path algorithms to identify criminal associations,” In Proceedings of the 2002 Annual National Conference on Digital Government Research (Los Angeles, California, May 19 - 22, 2002). vol. 129. Digital Government Society of North America, pp. 1-7.
54. Yu, C.; Leon-Garcia, A.; Foster, I., "Toward an Autonomic Service Management Framework: A Holistic Vision of SOA, AON, and Autonomic Computing," Communications Magazine, IEEE , vol.46, no.5, pp.138-146, May 2008
55. Yu, T.; Lin, K.J., “Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints,” Proc. Int’l Conf. Service-Oriented Computing (ICSOC ’05), 2005, pp. 130-143.

56. Zaiane, O.R.; Man, X.; Jiawei, H., "Discovering Web access patterns and trends by applying OLAP and data mining technology on Web logs," Research and Technology Advances in Digital Libraries, 1998. ADL 98. Proceedings. IEEE International Forum on , vol., no., pp.19-29, 22-24 Apr 1998
57. Zeng, L.; Benatallah, B.; Ngu, H. H. A.; Dumas, M.; Kalagnanam, J.; Chang, H., "QoS-Aware Middleware for Web Services Composition," IEEE Trans. Software Eng. 30(5): 311-327 (2004)
58. Zhang, N.; Qiu, X.S.; Meng, L.M., "A SLA-Based Service Process Management Approach for SOA," Communications and Networking in China, 2006. ChinaCom '06. First International Conference on, pp.1-6, 25-27 Oct. 2006
59. Zhao, Q.; Hoi, S. C. H.; Liu, T. Y.; Bhowmick, S. S.; Lyu, M. R.; Ma, W. Y., "Time-dependent semantic similarity measure of queries using historical click-through data," in WWW '06: Proceedings of the 15th international conference on World Wide Web. New York, NY, USA: ACM, 2006, pp. 543-552.
60. Zheng, X.; Yan, Y., "An Efficient Syntactic Web Service Composition Algorithm Based on the Planning Graph Model," Web Services, 2008. ICWS '08. IEEE International Conference on, pp.691-699, 23-26 Sept. 2008
61. Zheng, Z.; Lyu, M.R., "A QoS-Aware Middleware for Fault Tolerant Web Services," Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on , vol., no., pp.97-106, 10-14 Nov. 2008

62. Zhou, C.; Chia, L.T.; Lee, B.S., "QoS measurement issues with DAML-QoS ontology," e-Business Engineering, 2005. ICEBE 2005. IEEE International Conference on, pp.395-402, 12-18 Oct. 2005

63. Zhou, N.; Zhang, L., "A Graph Theory Based Impact and Completion Analysis Framework and Applications for Modeling SOA Solution Components," Services Computing, 2008. SCC '08. IEEE International Conference on, pp.145-154, 7-11 July 2008

Appendix A

EXTERNAL FILES GENERATED BY SR TOOL

1- "Simudat.txt" contains the new rankings based on the combination of QUV and SUV and different values of α and β. Old rankings are based on purely overall QUV scores. We also include rankings based on the combination of QUV and PUV in order to show the different performances of combining PageRank or Service Rank with the QoS values. "Simudat.txt" also includes the percentage of each composition usages in our simulation experiment. The percentages are kept fixed when we evaluate the effect of changing α and β values. This is the main result file.

2- "Graphc.gv" consists of visualization script for visualizing the web services registry graph by importing it to the Graphviz tool.

3- "historyl#.txt" includes the history usage data which saves the required information illustrated in Table 1 and Table 2 for each invocation. The "#" sign represents the experiment number, e.g. "historyl1.txt" for experiment 1.

4- "Input.doc" consists of web registry graph input data which is defined as a adjacency matrix to represent web services and their relations. This input data file is assumed to be obtained from a web services registry and it can be entered manually, automatically or from a file.

5- "Pagerank.txt" contains PageRank values for all of the web services.

6- “Pathff.txt” includes compositions generated by the SR tool and it shows the node numbers in each path. “-1” indicates the end of a path.

7- “Pathfile.txt” is another version of “Pathff.txt” file (in a different format).

8- “Servicernk1.txt” consists of Service Rank values for all of the web services.

9- “Soluq.txt” includes compositions solutions with their QoS values (the number of results are limited in the run time in case of having too many paths, e.g. maximum 200 compositions).

10- “Soluqsorted.txt” contains sorted compositions based on the overall QUV score with all of their nodes.

11- “Solusorted#.txt” consists of sorted compositions based on the combination of normalized SUV and QUV scores with different α and β values included in “Simudat.txt”.

12- Table.txt: contains QoS and PageRank values of each node

Appendix B

SAMPLE RESULTS GENERATED BY SR TOOL AND GRAPHS GENERATED BY GRAPHVIZ TOOL

```
digraph abstract {
graph [bgcolor=white];
edge [color=black];
graph[page="11,11",size="11,11",ratio=fill,center=1];
1->4;
1->6;
1->7;
1->10;
1->11;
1->14;
1->16;
1->17;
1->18;
1->21;
1->22;
1->24;
1->25;
1->27;
1->29;
1->30;
2->1;
2->3;
2->4;
2->6;
2->11;
2->14;
2->15;
2->16;
2->19;
2->20;
2->22;
2->24;
2->28;
2->29;
2->30;
```

Figure B.1 – Part of visualization script generated by SR tool for experiment #1

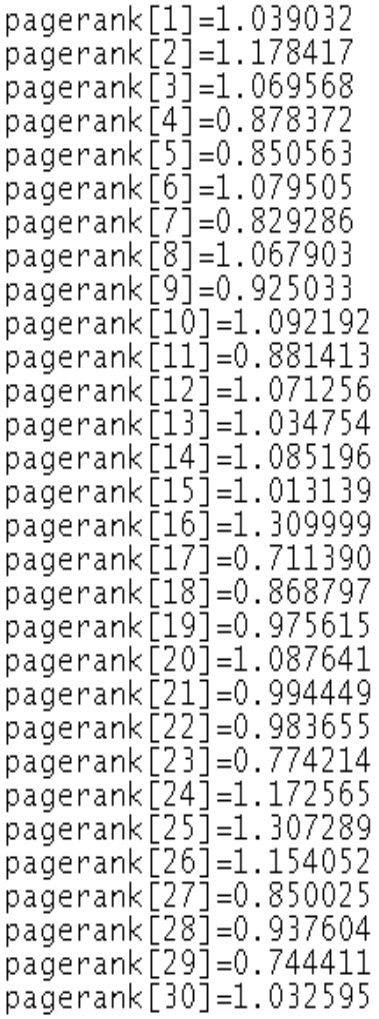


Figure B.2 – PageRank scores for experiment #1

```
servicerank[1]=0.150137
servicerank[2]=0.157923
servicerank[3]=0.150137
servicerank[4]=0.155269
servicerank[5]=0.150109
servicerank[6]=0.150134
servicerank[7]=0.150100
servicerank[8]=0.159557
servicerank[9]=0.150118
servicerank[10]=0.150141
servicerank[11]=0.150112
servicerank[12]=0.150146
servicerank[13]=0.160029
servicerank[14]=0.150142
servicerank[15]=0.150130
servicerank[16]=0.150175
servicerank[17]=0.161593
servicerank[18]=0.150110
servicerank[19]=0.150124
servicerank[20]=0.150148
servicerank[21]=0.150134
servicerank[22]=0.150129
servicerank[23]=0.150092
servicerank[24]=0.150154
servicerank[25]=0.150182
servicerank[26]=0.150155
servicerank[27]=0.162090
servicerank[28]=0.150125
servicerank[29]=0.150089
servicerank[30]=0.175055
```

Figure B.3– Service Rank scores for experiment #1

```
1 30 -1
1  7  30  -1
1  10  30  -1
1  16  30  -1
1  17  30  -1
1  21  30  -1
1  24  30  -1
1  25  30  -1
1  27  30  -1
1  29  30  -1
1  4  10  30  -1
1  4  12  30  -1
1  4  13  30  -1
1  4  15  30  -1
1  4  16  30  -1
1  4  17  30  -1
1  4  25  30  -1
1  4  27  30  -1
1  6  7  30  -1
1  6  12  30  -1
1  6  16  30  -1
1  6  17  30  -1
1  6  21  30  -1
1  6  24  30  -1
1  6  25  30  -1
1  6  27  30  -1
1  7  9  30  -1
1  7  10  30  -1
1  7  15  30  -1
1  7  21  30  -1
1  7  25  30  -1
1  7  27  30  -1
1  7  29  30  -1
1  10  7  30  -1
1  10  9  30  -1
1  10  12  30  -1
1  10  17  30  -1
1  10  21  30  -1
1  10  24  30  -1
```

Figure B.4– Part of path file showing some of the compositions in experiment #1

------Solution Number------	----------Scalability----------	----------Accuracy----------	-----Stabilitychangecycle-------
174	0.368200	2.636400	2.831600
129	0.601400	2.072000	2.679600
18	0.601400	1.729300	2.180300
185	0.368200	2.600200	3.306000
9	0.601400	1.027400	1.588600
103	0.241700	1.847300	2.361400
66	0.368200	1.934500	2.239900
163	0.601400	2.329300	2.555200
1	0.601400	0.520900	0.919000
142	0.368200	2.615600	3.008800
194	0.268100	2.402000	2.996000
171	0.140500	2.577600	2.857200
176	0.140500	2.577600	2.857200
16	0.368200	2.098400	2.368400
151	0.241700	2.441100	2.989700
133	0.368200	3.047200	2.876100
189	0.268100	2.433500	2.789600
131	0.241700	2.991500	2.791200
158	0.601400	2.360800	2.348800
147	0.368200	2.292300	2.822800
175	0.368200	2.292300	2.822800
5	0.368200	1.396500	1.776700
81	0.116400	1.467900	2.189800
180	0.601400	2.262600	2.911500
26	0.601400	1.573800	2.327800
178	0.078300	2.128100	2.806100
159	0.210800	1.846000	2.855800
63	0.140500	1.875700	2.265500
68	0.140500	1.875700	2.265500
32	0.210800	1.050600	2.558800
118	0.368200	2.946500	2.822200
143	0.523100	2.246500	2.820700
181	0.210800	1.747800	3.418500
41	0.601400	1.794200	1.682000
37	0.368200	2.345300	2.284400
108	0.223300	1.770100	2.504800
177	0.116400	2.225500	2.866400
121	0.601400	2.103500	2.473200
27	0.210800	1.364000	2.662000

Figure B.5 – Part of sorted compositions based on QUV in experiment #1

------InvocationTime-------	------User#------	------ContractDr------	------ContractStD------
2009/05/26 04:16	1	7 Months	2009/01/24
2008/03/01 22:13	2	5 Months	2007/11/18
2008/11/26 16:51	3	10 Months	2008/05/11
2008/08/21 15:44	4	2 Months	2008/07/18
2009/07/19 13:43	5	9 Months	2009/02/14
2009/00/04 08:48	6	2 Months	2008/11/02
2008/09/28 13:48	7	6 Months	2008/05/13
2008/06/15 21:28	8	11 Months	2007/11/01
2009/03/02 06:59	9	2 Months	2009/02/00
2008/05/00 10:32	10	11 Months	2007/11/14
2008/07/26 09:03	11	10 Months	2008/02/12
2009/02/29 05:25	12	3 Months	2009/00/24
2009/03/17 03:31	13	6 Months	2008/11/17
2008/09/18 23:48	14	1 Months	2008/09/03
2008/00/11 15:07	15	2 Months	2007/10/27
2009/06/11 14:48	16	6 Months	2009/02/25
2008/03/28 08:23	17	2 Months	2008/02/16
2008/08/17 19:37	18	7 Months	2008/04/29
2009/03/06 06:46	19	3 Months	2009/01/02
2008/01/11 17:20	20	6 Months	2007/09/08
2008/04/09 22:20	21	3 Months	2008/02/22
2008/09/05 18:42	22	11 Months	2008/01/03
2008/04/18 11:24	23	1 Months	2008/03/27
2008/04/04 07:07	24	1 Months	2008/03/12
2009/03/08 08:16	25	11 Months	2008/09/09
2008/01/00 05:48	26	2 Months	2007/11/23
2009/05/27 14:18	27	6 Months	2009/01/24
2009/01/03 23:25	28	6 Months	2008/10/03
2009/06/28 20:12	29	7 Months	2009/03/10
2008/02/27 17:17	30	11 Months	2007/07/11
2008/06/01 12:42	31	8 Months	2008/00/10
2008/02/20 01:28	32	4 Months	2007/11/20
2007/11/23 13:10	33	10 Months	2007/05/13
2008/02/23 17:27	34	3 Months	2008/00/22
2008/09/17 05:52	35	11 Months	2008/02/05
2009/04/28 11:06	36	9 Months	2008/11/07

Figure B.6– Part of history log data for experiment #1 (for solution #174)

------NormalizedSUV------	------NormalizedQUV------	------FinalScore------	------Rank#----
0.973894	0.932084	0.940446	1
0.597597	1.000000	0.919519	2
0.696961	0.947330	0.897256	3
0.663504	0.952797	0.894939	4
0.780456	0.920815	0.892743	5
1.000000	0.861484	0.889187	6
0.497675	0.927198	0.841293	7
0.948297	0.806570	0.834915	8
0.356214	0.940575	0.823703	9
0.677763	0.834865	0.803445	10
0.662770	0.811895	0.782070	11
0.662770	0.811895	0.782070	12
0.371729	0.874372	0.773844	13
0.372296	0.855902	0.759181	14
0.356876	0.856869	0.756871	15
0.357789	0.852526	0.753579	16
0.357789	0.852526	0.753579	17
0.356741	0.831014	0.736160	18
0.498311	0.778504	0.722466	19
0.243346	0.829446	0.712226	20
0.242780	0.817324	0.702415	21
0.480696	0.752444	0.698094	22
0.480696	0.752444	0.698094	23
0.496981	0.745992	0.696190	24
0.357138	0.777035	0.693056	25
0.861923	0.633284	0.679012	26
0.242622	0.784534	0.676152	27
0.001574	0.832325	0.666175	28
0.372234	0.737690	0.664599	29
0.355729	0.741588	0.664417	30
0.479386	0.705790	0.660509	31
0.534555	0.688702	0.657873	32
0.001566	0.818491	0.655106	33
0.336240	0.724530	0.646872	34
0.035116	0.796625	0.644323	35
0.479112	0.665683	0.628369	36

Figure B.7 - Part of sorted compositions (α=0.2 and β=0.8) based on combined normalized SUV and QUV for experiment #1

------NormalizedPUV------	------NormalizedQUV------	------PUVQUVScore------	------Rank#----
0.243000	1.000000	0.848600	1
0.263843	0.952797	0.815007	2
0.326003	0.920815	0.801853	3
0.264949	0.932084	0.798657	4
0.176459	0.947330	0.793156	5
0.494455	0.861484	0.788078	6
0.219690	0.927198	0.785696	7
0.092687	0.940575	0.770997	8
0.521872	0.829446	0.767931	9
0.340527	0.852526	0.750126	10
0.340527	0.852526	0.750126	11
0.466652	0.817324	0.747190	12
0.223252	0.874372	0.744148	13
0.278472	0.855902	0.740416	14
0.332255	0.832325	0.732311	15
0.337440	0.818491	0.722281	16
0.163575	0.856869	0.718210	17
0.438839	0.784534	0.715395	18
0.353496	0.796625	0.707999	19
0.179092	0.831014	0.700630	20
0.362805	0.778504	0.695364	21
0.447911	0.752444	0.691537	22
0.447911	0.752444	0.691537	23
0.048017	0.834865	0.677496	24
0.136520	0.811895	0.676820	25
0.136520	0.811895	0.676820	26
0.483526	0.724530	0.676329	27
0.686680	0.665269	0.669551	28
0.238663	0.777035	0.669361	29
0.880602	0.610990	0.664912	30
0.093693	0.806570	0.663995	31
0.207881	0.764952	0.653538	32
0.507242	0.688702	0.652410	33
0.266328	0.737690	0.643418	34
0.442446	0.683448	0.635248	35
0.444053	0.678534	0.631637	36
0.427910	0.677199	0.627341	37
0.130982	0.745992	0.622990	38
0.387226	0.675123	0.617544	39

Figure B.8 - Part of sorted compositions (α=0.2 and β=0.8) based on combined normalized PUV and QUV for experiment #1

----------NodeNumber----------	----------Scalability----------	----------Accuracy----------	-----Stabilitychangecycle-------
1	0.601400	0.266700	0.787900
2	0.715700	0.538000	0.463200
3	0.987600	0.400700	0.636900
4	0.660300	0.701900	0.591700
5	0.718900	0.848100	0.453800
6	0.794300	0.546400	0.739200
7	0.210800	0.023200	0.970200
8	0.643500	0.342700	0.499300
9	0.241700	0.819900	0.772800
10	0.654200	0.948800	0.507700
11	0.973400	0.735300	0.299800
12	0.523100	0.517200	0.640400
13	0.794400	0.193900	0.454400
14	0.903100	0.398400	0.706200
15	0.674800	0.600000	0.374900
16	0.140500	0.479200	0.488800
17	0.368200	0.875600	0.857700
18	0.884100	0.749500	0.328600
19	0.605100	0.501800	0.937600
20	0.268100	0.672700	0.815700
21	0.116400	0.127100	0.498000
22	0.111000	0.524100	0.543100
23	0.753000	0.452400	0.082200
24	0.078300	0.029700	0.437700
25	0.437100	0.447300	0.076900
26	0.789100	0.836700	0.114100
27	0.630000	0.506500	0.669600
28	0.392100	0.635900	0.263200
29	0.223300	0.429300	0.813000
30	0.933200	0.254200	0.131100

Figure B.9 – Part of web services' data for experiment #1

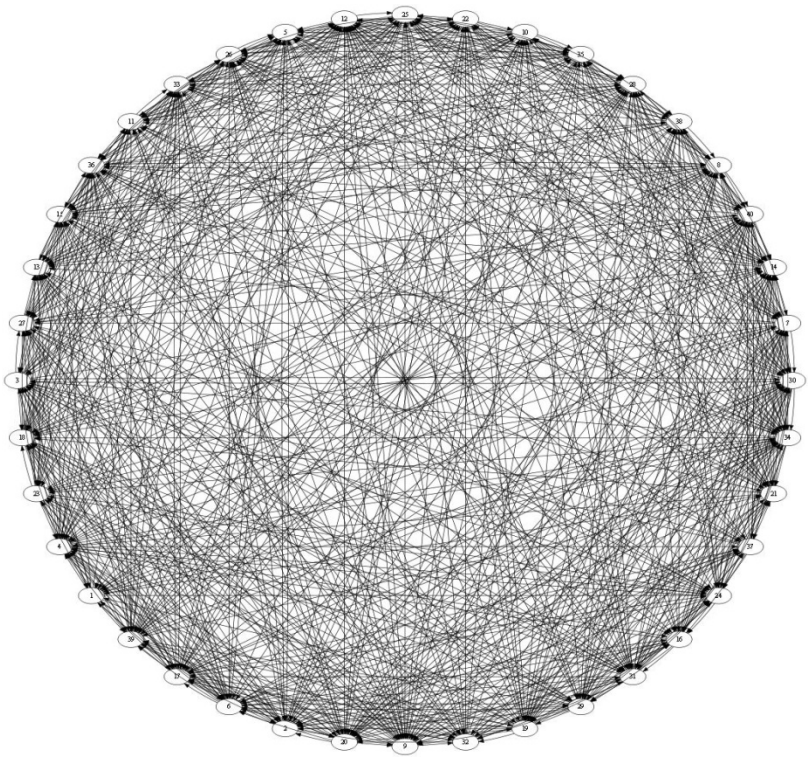


Figure B.10 – Web services registry graph for experiment #2

```
pagerank[1]=0.677118
pagerank[2]=0.972227
pagerank[3]=0.865146
pagerank[4]=1.076923
pagerank[5]=0.963672
pagerank[6]=1.136163
pagerank[7]=0.944947
pagerank[8]=0.974495
pagerank[9]=1.265027
pagerank[10]=0.876129
pagerank[11]=0.981966
pagerank[12]=1.324997
pagerank[13]=1.084904
pagerank[14]=1.177224
pagerank[15]=1.105028
pagerank[16]=1.082498
pagerank[17]=1.026888
pagerank[18]=1.046941
pagerank[19]=0.811644
pagerank[20]=1.062094
pagerank[21]=0.914297
pagerank[22]=0.881318
pagerank[23]=0.827826
pagerank[24]=1.145510
pagerank[25]=0.897013
pagerank[26]=1.044989
pagerank[27]=1.024364
pagerank[28]=1.004215
pagerank[29]=0.941344
pagerank[30]=1.178188
pagerank[31]=0.991273
pagerank[32]=0.835286
pagerank[33]=0.992545
pagerank[34]=1.099749
pagerank[35]=0.903880
pagerank[36]=1.068831
pagerank[37]=0.811592
pagerank[38]=1.047177
pagerank[39]=0.975130
pagerank[40]=0.960172
```

Figure B.11– PageRank scores for experiment #2

```
servicerank[1]=0.150081
servicerank[2]=0.150127
servicerank[3]=0.156581
servicerank[4]=0.150142
servicerank[5]=0.150124
servicerank[6]=0.153388
servicerank[7]=0.150123
servicerank[8]=0.150125
servicerank[9]=0.150168
servicerank[10]=0.150111
servicerank[11]=0.150123
servicerank[12]=0.150176
servicerank[13]=0.150143
servicerank[14]=0.154764
servicerank[15]=0.150141
servicerank[16]=0.150137
servicerank[17]=0.157562
servicerank[18]=0.150137
servicerank[19]=0.156266
servicerank[20]=0.150141
servicerank[21]=0.150119
servicerank[22]=0.150107
servicerank[23]=0.150101
servicerank[24]=0.154261
servicerank[25]=0.150113
servicerank[26]=0.150138
servicerank[27]=0.150136
servicerank[28]=0.150127
servicerank[29]=0.150117
servicerank[30]=0.150156
servicerank[31]=0.156384
servicerank[32]=0.150105
servicerank[33]=0.150129
servicerank[34]=0.150145
servicerank[35]=0.150115
servicerank[36]=0.150139
servicerank[37]=0.150092
servicerank[38]=0.150135
servicerank[39]=0.159777
servicerank[40]=0.168627
```

Figure B. 12 – Service Rank scores for experiment #2

```
1 40 -1
1  7  40  -1
1  24  40  -1
1  27  40  -1
1  38  40  -1
1  39  40  -1
1  3  2  40  -1
1  3  5  40  -1
1  3  9  40  -1
1  3  12  40  -1
1  3  18  40  -1
1  3  22  40  -1
1  3  24  40  -1
1  3  27  40  -1
1  3  28  40  -1
1  3  29  40  -1
1  3  31  40  -1
1  4  2  40  -1
1  4  5  40  -1
1  4  7  40  -1
1  4  8  40  -1
1  4  12  40  -1
1  4  22  40  -1
1  4  24  40  -1
1  4  27  40  -1
1  4  28  40  -1
1  4  37  40  -1
1  4  39  40  -1
1  6  5  40  -1
1  6  18  40  -1
1  6  27  40  -1
1  6  28  40  -1
1  6  29  40  -1
1  6  31  40  -1
1  6  37  40  -1
1  6  39  40  -1
1  7  5  40  -1
1  7  8  40  -1
1  7  9  40  -1
```

Figure B. 13 – Part of path file showing some of the compositions in experiment #2

------Solution Number------	----------Scalability----------	----------Accuracy----------	-----Stabilitychangecycle-------
56	0.148900	3.338800	2.453000
61	0.076000	3.025100	1.942500
57	0.148900	3.324100	1.865700
53	0.148900	2.564100	2.277100
20	0.002700	3.450100	2.555700
24	0.002700	3.403700	2.024200
71	0.148900	3.494900	2.068500
48	0.009700	3.048300	1.383100
1	0.148900	1.622900	1.118000
104	0.148900	2.969400	2.661300
33	0.148900	3.047900	1.986800
74	0.148900	3.044000	1.994500
55	0.148900	2.432800	2.415200
50	0.009700	2.814100	2.001800
131	0.002700	3.431500	2.482400
130	0.002700	3.601000	2.857100
101	0.148900	2.605400	2.418700
97	0.076000	2.246900	2.275200
6	0.076000	2.236700	1.404300
36	0.076000	3.153400	1.606100
194	0.148900	2.936900	3.086400
99	0.148900	2.179800	1.988100
134	0.002700	3.554600	2.325600
188	0.148900	3.048000	2.624500
41	0.148900	3.509600	2.655800
193	0.148900	3.171100	2.467700
100	0.148900	2.831600	2.230800
19	0.002700	2.628800	1.938300
28	0.002700	3.104700	2.101000
29	0.148900	2.677500	1.443400
89	0.140100	1.771000	2.112500
21	0.002700	3.280600	2.181000
40	0.148900	2.603600	2.618000
105	0.148900	2.954700	2.074000
94	0.140100	2.545900	2.198400
37	0.148900	2.720000	1.982600

Figure B. 14 – Part of sorted compositions based on QUV for experiment #2

------InvocationTime-------	------User#------	------ContractDr------	------ContractStD------
2008/00/08 00:47	1	9 Months	2007/07/20
2008/09/16 02:42	2	6 Months	2008/05/06
2008/00/04 18:41	3	5 Months	2007/09/10
2009/05/14 12:08	4	3 Months	2009/03/16
2009/00/00 20:40	5	3 Months	2008/10/05
2008/09/05 14:29	6	2 Months	2008/07/24
2008/05/17 17:57	7	12 Months	2007/08/24
2008/08/18 08:45	8	2 Months	2008/07/09
2009/03/01 20:41	9	12 Months	2008/06/23
2008/04/16 17:09	10	3 Months	2008/02/25
2008/05/26 22:27	11	10 Months	2007/11/26
2009/07/26 20:26	12	2 Months	2009/06/19
2009/00/02 18:25	13	11 Months	2008/06/08
2008/06/15 10:01	14	5 Months	2008/03/02
2008/07/04 10:35	15	11 Months	2007/11/28
2009/04/06 02:43	16	10 Months	2008/09/25
2009/00/22 01:38	17	7 Months	2008/07/29
2009/05/11 10:51	18	9 Months	2009/00/13
2008/00/27 04:56	19	7 Months	2007/08/04
2009/00/18 10:03	20	11 Months	2008/05/15
2008/11/13 11:00	21	12 Months	2008/04/25
2008/09/19 05:06	22	12 Months	2008/02/06
2008/11/26 07:27	23	4 Months	2008/09/16
2009/05/18 10:24	24	6 Months	2009/01/24
2009/06/00 01:58	25	6 Months	2009/02/09
2008/02/28 18:15	26	5 Months	2007/11/11
2008/06/03 11:25	27	11 Months	2007/10/23
2008/11/27 06:59	28	8 Months	2008/07/00
2008/09/08 08:47	29	7 Months	2008/04/06
2008/00/12 10:16	30	1 Months	2007/11/22
2008/03/05 05:58	31	7 Months	2007/10/00
2009/00/12 15:16	32	6 Months	2008/09/12
2009/00/28 12:34	33	6 Months	2008/08/26
2009/04/17 12:48	34	2 Months	2009/03/05
2008/02/11 13:38	35	11 Months	2007/07/05
2008/04/08 15:56	36	3 Months	2008/02/22

Figure B. 15 – Part of history log data for experiment #2 (for solution #56)

------NormalizedSUV------	------NormalizedQUV------	------FinalScore------	------Rank#------
0.898595	0.984594	0.958794	1
0.714420	1.000000	0.914326	2
0.970418	0.817051	0.863061	3
1.000000	0.742982	0.820088	4
0.609204	0.902686	0.814642	5
0.392295	0.864962	0.723162	6
0.679573	0.739101	0.721243	7
0.392833	0.800911	0.678488	8
0.536738	0.738049	0.677656	9
0.508608	0.747663	0.675946	10
0.325266	0.810973	0.665261	11
0.219887	0.854129	0.663856	12
0.331277	0.800143	0.659483	13
0.509278	0.706746	0.647506	14
0.614145	0.653308	0.641559	15
0.218901	0.821449	0.640684	16
0.447616	0.720208	0.638431	17
0.219875	0.817419	0.638156	18
0.218575	0.804575	0.628775	19
0.508491	0.671697	0.622735	20
0.531789	0.658036	0.620162	21
0.172749	0.808493	0.617770	22
0.324117	0.722853	0.603232	23
0.002780	0.858733	0.601947	24
0.274039	0.723726	0.588820	25
0.253615	0.724665	0.583350	26
0.501584	0.607652	0.575832	27
0.080038	0.781172	0.570832	28
0.485128	0.587405	0.556722	29
0.079930	0.756065	0.553224	30
0.329354	0.642536	0.548582	31
0.220050	0.688662	0.548078	32
0.484348	0.572027	0.545723	33
0.330619	0.637144	0.545186	34
0.173116	0.704463	0.545059	35
0.219216	0.682152	0.543271	36

Figure B.16 - Part of sorted compositions (α=0.3 and β=0.7) based on combined SUV and QUV for experiment #2

------NormalizedPUV------	------NormalizedQUV------	------PUVQUVScore------	------Rank#----
0.642219	0.902686	0.824546	1
0.437735	0.984594	0.820536	2
0.702270	0.854129	0.808571	3
0.736001	0.817419	0.792994	4
0.873104	0.752076	0.788384	5
0.241525	1.000000	0.772457	6
0.461561	0.858733	0.739581	7
0.543877	0.821449	0.738177	8
0.434250	0.864962	0.735748	9
0.550891	0.800143	0.725367	10
0.539553	0.804575	0.725068	11
0.528334	0.808493	0.724446	12
0.523920	0.800911	0.717814	13
0.702766	0.720208	0.714975	14
0.729664	0.688662	0.700963	15
0.606464	0.724665	0.689205	16
0.568882	0.739101	0.688035	17
0.554678	0.738049	0.683038	18
0.442266	0.781172	0.679500	19
0.538568	0.723726	0.668179	20
0.328970	0.810973	0.666372	21
0.511429	0.726138	0.661725	22
0.555131	0.704463	0.659664	23
0.601003	0.682152	0.657807	24
0.524418	0.711876	0.655639	25
0.413896	0.756065	0.653414	26
0.787068	0.595321	0.652845	27
0.497786	0.706746	0.644058	28
0.633448	0.646493	0.642580	29
0.396518	0.747663	0.642319	30
0.484034	0.710044	0.642241	31
1.000000	0.480783	0.636548	32
0.497023	0.695615	0.636037	33
0.608126	0.641005	0.631141	34
0.666570	0.603511	0.622429	35
0.666570	0.603511	0.622429	36

Figure B.17- Part of sorted compositions (α=0.3 and β=0.7) based on combined PUV and QUV for experiment #2

----------NodeNumber----------	----------Scalability----------	----------Accuracy----------	-----Stabilitychangecycle-------
1	0.325400	0.879900	0.183500
2	0.957500	0.152800	0.620900
3	0.237700	0.150900	0.301400
4	0.002700	0.868000	0.696700
5	0.730700	0.137900	0.123600
6	0.901000	0.916700	0.201800
7	0.408800	0.959200	0.741000
8	0.989300	0.789700	0.366300
9	0.958700	0.563500	0.554200
10	0.616700	0.819200	0.599700
11	0.941600	0.655500	0.684400
12	0.097000	0.507300	0.474900
13	0.740100	0.042800	0.196900
14	0.703200	0.484500	0.838800
15	0.009700	0.512600	0.055600
16	0.547500	0.129200	0.445200
17	0.420200	0.788400	0.538200
18	0.919500	0.021500	0.759000
19	0.392400	0.927500	0.796800
20	0.761900	0.843100	0.789500
21	0.726100	0.694900	0.172800
22	0.898200	0.305100	0.305800
23	0.647800	0.924300	0.244800
24	0.905700	0.912800	0.209500
25	0.340000	0.608000	0.627300
26	0.471600	0.023400	0.412700
27	0.002100	0.121400	0.086300
28	0.550200	0.673300	0.912600
29	0.496200	0.508300	0.667000
30	0.622300	0.179200	0.162700
31	0.335900	0.678600	0.828200
32	0.290200	0.800200	0.074000
33	0.140100	0.010200	0.870900
34	0.199800	0.419000	0.746500
35	0.370100	0.523100	0.659100
36	0.581800	0.813600	0.902100
37	0.858000	0.014700	0.484300
38	0.925800	0.154000	0.156400
39	0.076000	0.613800	0.286300
40	0.148900	0.743000	0.934500

Figure B.18 - Part of web services' data table for experiment #2

```
                                        *****simulation=4*****
alpha=0.300000 beta=0.700000
rankings based on QUV:
 solution# 56 --> rank: 1
 solution# 61 --> rank: 2
 solution# 57 --> rank:3
 solution# 36 --> rank: 20
 solution# 194 --> rank:21
 average ranking is: 9.400000
alpha*PUV and beta*QUV rankings:
 solution# 57 --> rank: 1
 solution# 61 --> rank:2
 solution# 56 --> rank: 6
 solution# 36 --> rank: 17
 solution# 194 --> rank: 18
average of used solutions rankings based on alpha*PUV and beta*QUV: 8.800000
alpha*SUV and beta*QUV rankings:
 solution# 61 --> rank: 1
 solution# 56 --> rank:2
 solution# 57 --> rank: 5
 solution# 36 --> rank: 7
 solution# 194 --> rank: 9
average of used solutions rankings based on alpha*SUV and beta*QUV: 4.800000
number of users is: 1000
 percentage of users which choose the first ranked composition based on QUV (solution number 56)is: 20.000000
 Percentage of users which choose the second ranked composition based on QUV (solution number 61)is: 20.000000
 Percentage of users which choose the third ranked composition based on QUV (solution number 57)is: 10.000000
 Percentage of users which choose the 20 th ranked composition based on QUV (solution number 36)is: 34.000000
 Percentage of users which choose the 21 th ranked composition based on QUV (solution number 194) is: 16.000000

                                        *****simulation=5*****
alpha=0.400000 beta=0.600000
rankings based on QUV:
```

Figure B. 19 - Part of simulation results with rankings for Experiment #2

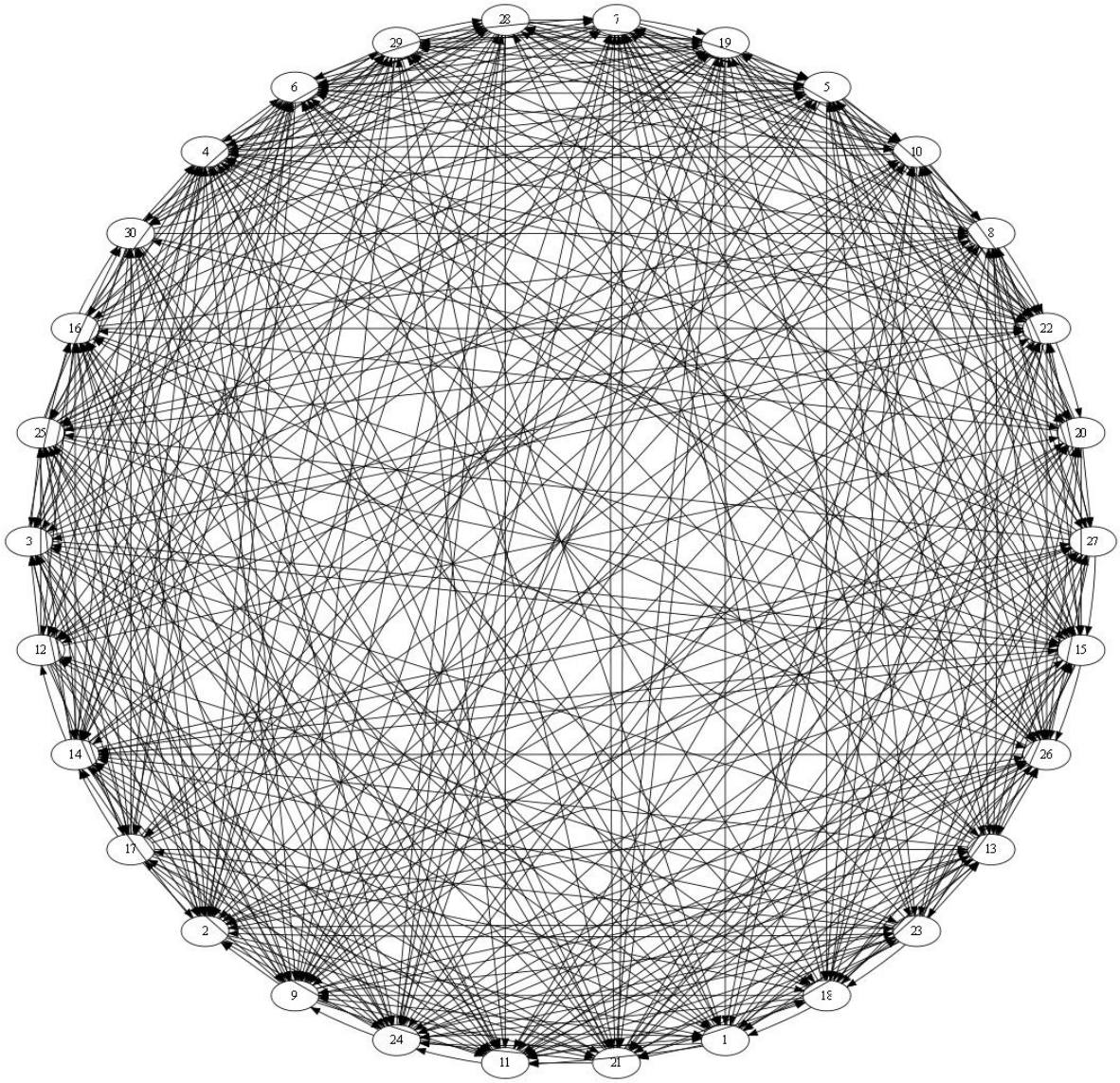


Figure B. 20 – Web services registry graph for experiments #3 and #4