1-1-2012

# An Efficient Qos-Based Ranking Model for Web Service Selection with Consideration \of User's Requirement

Anita Mohebi
*Ryerson University*

# AN EFFICIENT QOS-BASED RANKING MODEL FOR WEB SERVICE SELECTION WITH CONSIDERATION OF USER'S REQUIREMENT

by

Anita Mohebi

B.Sc., Applied Mathematics, Isfahan University

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, 2012

# AUTHOR'S DECLARATION

ANITA MOHEBI

# AN EFFICIENT QOS-BASED RANKING MODEL FOR WEB SERVICE SELECTION WITH CONSIDERATION OF USER'S REQUIREMENT

Anita Mohebi

Master of Science, Computer Science, 2012

Ryerson University

## ABSTRACT

The power of Web services to address the incompatibility issue of standalone systems, has led them to play a major role in business application development. Adopting an efficient and effective method to locate and select desired services among thousands of available candidates is an important task in the service-oriented computing. As part of a Web service discovery system, the ranking process enables users to locate their desired services more effectively. Many of the existing approaches ignore the role of user's requirements which is an important factor in the ranking process. In this thesis we enhance a vector-based ranking method by considering user's requirements. The vector-based model is chosen because of its simplicity and high efficiency. We evaluate all Web services in terms of their similarity degrees to the optimal or the best available values of each quality attribute, and penalize the services that fail to meet the user's requirements. Through our extensive experiments using real datasets, we compare the improved algorithm with other approaches to evaluate it in terms of efficiency (the execution time to return the result) and quality of the results (accuracy).

# ACKNOWLEDGEMENTS

This thesis would not be finished without the support of my companions during my study. Firstly I would like to thank my supervisor Dr. Cherie Ding, for her consideration and valuable assistance to conduct this research. She patiently supported me in different situations, helped me to overcome the obstacles and guided me through all my academic requirements.

Furthermore, I would like to thank the computer science department for supporting me and providing me with all the requirements to fulfill this research. I also would like to express my deepest respect and gratitude to Dr. Sadeghian for all his guidance and support.

Moreover my appreciation goes to my husband for being a patient companion during my study. Without his constant encouragement this thesis could not be finished. Finally I would like to express my love and respect to my parents and acknowledge their support to accomplish this work. I could not pursue my study without their invaluable support and encouragement.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ACRONYMS

| | |
|---|---|
| AHP | Analytic Hierarchy Process |
| BF | Borda Fuse |
| BF_Q | Borda Fuse with Query |
| CP | Constraint  Programming |
| CSO | Constraint Satisfaction Optimization |
| CSP | Constraint Satisfaction Problem |
| DS | Distance Based |
| DS_I | Improved DS |
| HTTP | Hypertest Transfer Protocol |
| IR | Information Retrieval |
| MCDA | Multi-Criteria Decision Analysis |
| MCDM | Multi-Criteria Decision Making |
| MIP | Mixed Integer Programming |
| NFP | Non-Functional Properties |
| OPL | Optimization Programming Language |
| OWL-Q | Ontology Web Language for Query Language |
| OWL-S | Ontology Web Language for Semantic Web services |
| PTF | Public Transportation Facility |
| QoS | Quality of Service |
| QWS | Quality Web Service |
| SAW | Simple Additive Weighting |
| SFD | Spearman Footrule Distance |
| SFS | Sort -Filter- Skyline |
| SOA | Service Oriented Architecture |
| UDDI | Universal Description, Definition, and Integration |
| VSM | Vector Space Model |
| WSDL | Web Service Description Language |
| XML | Extensible Markup Language |

# CHAPTER 1

# INTRODUCTION

## 1.1. Background

The growing number of business applications in distributed systems has resulted in the increasing demand of communication between business modules. In context of the business community, Service Oriented Architecture (SOA) was proposed based on the idea that to provide a solution for a large problem in a more effective way, the required process can be decomposed into a collection of smaller, but related parts [1]. The most common way to implement SOA is through Web services. According to W3C (World Wide Web Consortium), a Web service is defined as a software module which is implemented through standard XML-based technologies such as WSDL and SOAP. With the increasing number of Web services, discovering and selecting best services to fulfill a required task is becoming more important. In order to search and invoke Web services based on user's requirements, first all functional services need to be advertised by their providers in a public UDDI (Universal Description, Discovery, and Integration) registry [2]. Service providers publish descriptions and properties of their Web services in a standard file, i.e. WSDL (Web Service Description Language). A WSDL file contains the information about data types, operations and the network location of the Web services. Then consumers create their queries and use a discovery facility or an agent to search UDDI and locate the set of Web services relevant to their desired requirements. Finally, consumers need to select and invoke one of the Web services among all retrieved results [3]. The steps of Web service discovery process are represented in Figure 1.1.

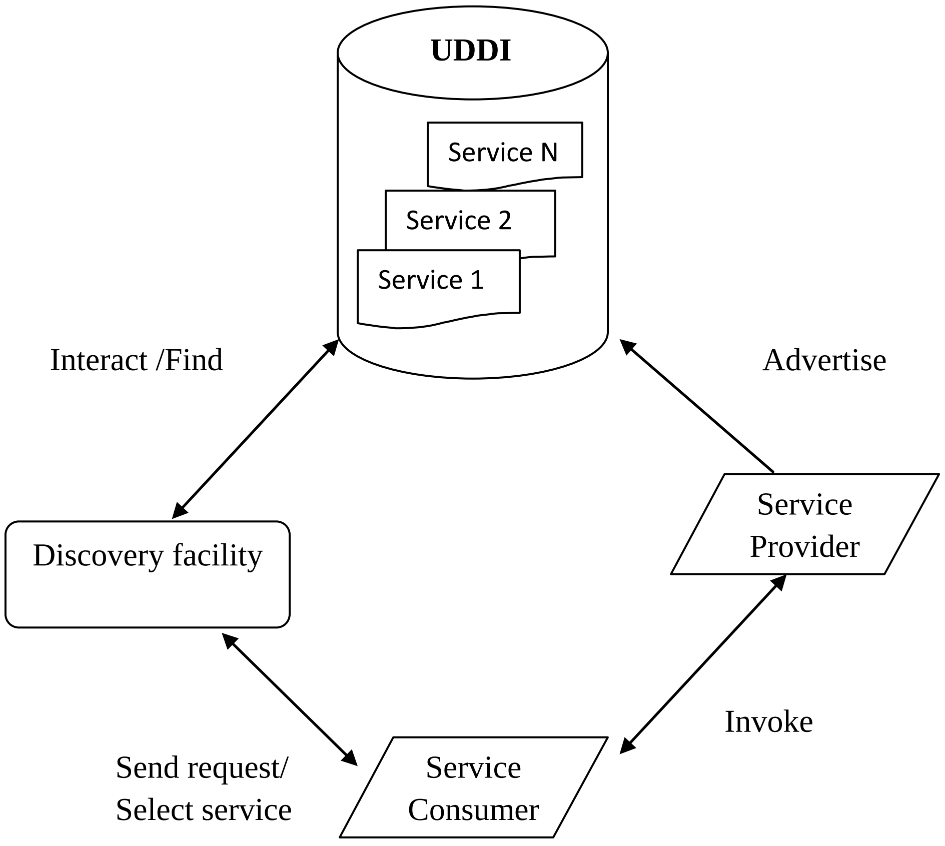

Figure1.1- Web Service discovery architecture

More and more web services with the similar functionality are made available on the Web. In order to locate and select the appropriate Web services, additional features, i.e. non-functional attributes or quality of Web services (QoS) such as response time, scalability, etc. are taken into consideration in the discovery and selection process.

With the increasing size of the UDDI registry, it is becoming more difficult to locate and retrieve all matched web services and present them to consumers. Furthermore, it is evident that the retrieved result contains more than one matched Web services that meet the functional and non-functional criteria. Therefore, it is essential to devise an efficient technique to measure the ranking relation order between the retrieved services based on user’s requirements on different QoS attributes. The process of ranking Web services is a dominant part of a Web service selection system, as it helps users select their desired service easily.

## 1.2. Problem Statement

Calculating the similarity degree between the user's request and a service is the fundamental step in the ranking process. There have been different approaches proposed for the problem of ranking Web services. These models mostly used different methods to compare all quality parameters of similar Web services with the optimal values for each QoS attribute. According to these approaches, the service with the maximum similarity degree to the optimal values will be returned as the result. There are some open issues that are not well supported by the current approaches. Firstly in some works they either used complicated data indexing methods in their query structure of the ranking process or compared all Web services in a pair-wise fashion which will result in a larger computation time. With the growing number of Web services with the similar functionality in a registry, the number of pair-wise comparisons will increase, which in turn makes the algorithms much slower.

Moreover, they mostly ignored the role of consumers in developing their models. To implement the discovery system, they only focused on the services with the optimal values, not the real constraints appeared in the query. In the final result list, users can only have access to those Web Services with minimum distance to the data items with optimal values. Regardless of user's actual requirements, the result of the algorithms is always the same. It is not rational to ignore consumer’s needs, while the main goal of the model is to respond to user's request.

Clients have different anticipations and definitions of an appropriate service, so without considering their demands, there is no guarantee to satisfy them.

Furthermore, most of the existing frameworks concentrated on a small and limited number of QoS attributes. In the majority of works, they considered variables of only numeric type and conducted their experiments on small-sized Web service repositories. In reality, there are various types of variables for QoS attributes, which need to be considered to fulfill a desired task. Consumers prefer efficient methods capable of dealing with different types of constraints and large-sized Web service repositories.

## 1.3. Motivation and Objective

Among the results returned from the matching process of the current solutions, there might be some Web services that are good candidates with regards to some QoS attributes and user’s actual demand. They can be potentially good options if a requester cares more about some specific attributes of services. But they are ranked very low as they do not attain enough scores compared to optimal values. Depending on the user's preferences they might also be willing to relax some of their requirements. Suppose that a user has a request for a travel Web service with response time less than 5 ms, availability greater than 90% and cost less than $50. The existing complicated algorithms only return those services with the optimal values on all three attributes. Suppose service A has response time: 3 ms, availability: 95% and cost $40, and service B has response time: 4 ms, availability 93% and price: $38. Using the current algorithms, service B might not be returned to the user as it gains smaller score than service A. Service A has better values for two attributes: "availability" and "response time". However service B can be a good candidate for some users who care more about the price of the service.

Another noticeable issue arises when user changes his demand. Using the current algorithms, even with different requirements, the result of the algorithms would always be the same, i.e. service A always appears on the top of the result list. Even for such a simple example, we can notice that how user's requirements can affect the output. As a result consumers' demands should play a pivotal role in the ranking process.

In real scenarios, value types of QoS attributes might be more diverse and complex. With different value types, we need to devise appropriate methods to compare their values in an effective way. We also need to consider a unified structure to combine various values of different QoS attributes into an overall ranking score to evaluate the ranking relation between the Web services.

Another problem is the existing complicated frameworks suffer from high computation time for processing a request. With the increasing number of published Web services, it is important to return the searching result to users fast. User's tolerance on slow response from the selection system is usually very low.

Our main goal is to develop a Web service ranking model in which, we not only consider the optimal values, but also exert user’s actual request and preferences in the model. We consider equal weights for both mentioned factors. Inspired from previous works, in this research we would like to use a simple and more straightforward method to rank retrieved Web services and achieve accurate results. By developing and optimizing a vector-based framework and considering user’s requests and preferences, we introduce a methodology to process the ranking task effectively.

In this thesis, we also consider one of the promising existing ranking algorithms to compare with the improved model in terms of the quality of the result and the efficiency. We believe that without applying a complicated methodology into the ranking process, which is vastly used in the current complex approaches, we can still have reliable results.

Considering efficiency and quality of the results, we also compare our model with a simple positional algorithm to show that our model is reliable and efficient. We compare the algorithms by using different number of Web services and different types of QoS attributes such as interval data and list which directly affect the processing time. Our experiment is based on a real QoS dataset.

## 1.4. Main Contributions

As the main contributions in this work, we provided an enhancement ranking algorithm based on a vector-based model which is capable of dealing with user's requirements and measuring the ranking relation between services and providing more accurate results efficiently. We also improved a rank aggregation based algorithm (i.e. Borda Fuse) to cover the user's requirement and provide more accurate results. Finally we compared the enhanced algorithms with one of the well-accepted skyline ranking algorithms (Sort-Filter-Skyline) with complex structure to show how they are more efficient on large sized datasets with a large number of attributes and different data types.

## 1.5. Outline of Thesis

The rest of the thesis is organized as follows:

Chapter 2 provides briefly the related information about selection and ranking methods of Web service discovery system and introduces some existing algorithms such as: Distance-based model, Vector-based approach, Mixed Integer Programming (MIP), etc.

Chapter 3 presents the baseline of three current well approved ranking algorithms such as: Borda Fuse, original Distance-based and Skyline operation methods. We discuss how they are applied to ranking Web services, and then we examine the common issues and problems that arise in ranking Web services in a selection system. Then we present our optimized Distance-based algorithm with consideration of user's requirements. We also enhance an improved Positional-based model by considering user’s requirements in the algorithm. We process user queries by applying equal weights on optimal-value based ranking scores and user-request based ranking scores to solve the current problems we mentioned earlier to some extent.

In Chapter 4, we explain the implementation of the optimized model and how we apply the new features in the original approach. In the experiment part, we discuss the model and the related results by using a real dataset including over 2000 Web services. Then we provide the evaluations of our developed ranking algorithm compared with the three current ranking methods in terms of their efficiencies and accuracies.

Finally, in Chapter 5 we conclude the thesis with a summary and discuss about the future work.

# CHAPTER 2
# RELATED WORK

Web services are becoming more popular in the business communities. There are more services coming to market to accomplish different tasks with the same functionality. Therefore it is crucial to use an effective method based on quality attributes to discover the best services in a desired ranked list according to consumers' requirements. Quality attributes of services (QoS) are considered as the discrimination factor for similar Web services. In general, most researchers classify Web service discovery and ranking methods into two different groups: syntactic and semantic approaches. In semantic methods, the ontology concepts are used in the discovery process, whereas in syntactic methods, the selection process is based on the syntactic information. In this chapter we will review some of the efforts under these two different categories. We also review some promising ranking methods originally presented in different fields, but are being used in the Web service selection systems.

## 2.1. Semantic Web Service Discovery

It is argued that as syntactic-based models only rely on syntactic information, so there is no guarantee to gain accurate results. Some researchers believe those models have low performance in locating desired results among retrieved services with the same functionality. On the contrary, in semantic-based methods, similarity between Web services and the request are computed based on some predefined QoS ontology, and their corresponding QoS metrics.

Bianchini et al. [4] introduced an ontology-based model which enriches the functional-based Web service discovery by considering both general and specific descriptions of QoS of Web services. In this model, they described each quality parameter by a name, a domain, a set of admissible values, and one or more measure units. They also considered a set of rules to convert the parameter values from a measure unit to another one. They used a semi-automated technique to find the relation between services and, then they assigned the weights to each relationship. In the next step, they formed a path from each service to its related services based on their semantic relationship. At the end they grouped the services in the similar sets. The drawback of their model is that they only considered the static quality attributes while there might be some dynamic parameters required to be taken into consideration.

Martin et al. [5] introduced OWL-S as a technology based on Ontology Web Language (OWL) which supports the structure for describing the characteristics of Web services published by providers and requested by user. Based on this model the results of the selection phase are more accurate and reliable and closer to consumer’s demand. However this method suffers from its lack of ability to deal with new metrics. Kritikos and Plexousakis [6] proposed a new ontology (OWL-Q) by extending OWL-S to overcome the issues related to new metrics. The structure is developed in a way to be able to adopt and extend new metrics with no need to modify the structure of the discovery algorithm. In other words, the advantage of this approach is to deal with dynamic quality properties.

Giallonardo and Zimeo [7] presented onQoS ontology to provide an automatic QoS-based service discovery. The model consists of three different layers: upper, middle and lower ontology. In the upper ontology the generic descriptions required for dealing with the query terms are provided. It provides the "words" required for answering the QoS queries. The middle layer describes the basic vocabulary list of different QoS features such as QoS metrics, scales and attributes. The vocabulary list consists of different parameters according to their conceptualizations. The last layer (lower layer) provides all specific definitions of a particular domain. Their algorithm includes two stages: atomic matching and aggregated matching. The output of the first phase is a list named *atomicMappingList* with information about required QoS metrics, advertisement QoS metrics and matching scores. The second phase recovers the template metrics that are not supported in the atomic matching layer.

Tomal et al. [8] described non-functional properties (NFP) of Web services by introducing an ontological model. To generate the ranked lists of the Web services based on user's preferences, they considered both semantic and multi-criteria types of ranking. They applied a reasoning engine to rank Web services according to NFP semantically. They set a tuple of QoS properties and their associated weights. The reasoning engine evaluates the logical rules used to model NFPs of services. To evaluate the ordering relations between the services, they used an aggregated score calculated for each service. The score is computed by summing up the normalized values of weighted NFP multiplied by their associated weight values. Then the scores are sorted to generate the final ranked list.

There is another category of methods based on Vector Space Model (VSM). In general, VSM is an algebraic model to represent different objects as vectors of identifiers. Each dimension represents a separate element. In the model introduced by Mola et al. [9], all retrieved Web services from the functional discovery process, are ranked according to a VSM model. They used OWL-S ontology to describe all properties of both queries and advertised Web services semantically. In this approach all requirements in the query and all properties of the retrieved Web services are considered as separate vectors. By using a defined semantic function, the similarity between the query vector and the vector of the offered Web services is measured. Those services which are more similar to the query will acquire higher position in the ultimate ranked list.

Zhou et al. [10] designed a QoS Ontology language in order to provide an agreement at the semantic level between different groups. Their Ontology consists of three layers: 1) the QoS profile layer designed to define a common class to accomplish the matching task. This step checks the retrieved services to see if they satisfy user's functional requirement; 2) the QoS property definition layer designed to define the properties and constraints based on a common language. They specified five classes for the QoS property’s domain: QoSCore, QoSInput, QoSOutput, QoSPrecondition, and QoSEffect. In this stage the services are compared based on their QoS values; 3) the metrics layer to define metrics. The metrics are categorized in two groups: AtomicMetrics and ComplexMetrics. The definition and properties of each AtomicMetric provide the important information to initiate the observer. ComplexMetircs are composed of AtomicMetrics or other ComplexMetrics. ComplexMetrics can be considered as a QoS metric which is able to describe any metric aggregation.

## 2.2. Syntactic Web Service Discovery

According to some researchers, semantic-based approaches suffer from massive human effort and complicated computational process resulting in the slow processing time. It is also assumed that there is no standard definition of ontology to use for different situations. People might use alternate concepts to define metrics. To address these issues, another category of service discovery approaches has been developed which is based on the syntactic information. It is believed by many researchers that syntactic-based models are more efficient than semantic-based approaches. There are various models with different methodologies introduced in this context.

Constraint Programming (CP) was proposed by Hentenryck and Saraswat [11] is one of the approaches in this context. According to the CP paradigm, the relations between variables are defined as constraints. Finding the optimal offers is an optimization problem which is well supported by the CP model. In the model proposed by Ruiz-Cortés et al. [12], CP was used for the following purposes: 1) to measure the conformance and consistency of the offers and demands to check if they have any internal contradictions, 2) to check if offered services and demanded services are similar, 3) to locate the optimal offer out of a set of similar services, 4) to support a two- way matchmaker which accepts condition both on published offers and demanded services. They used an OPL (optimization programming language) studio as the constraint solver to accomplish the mentioned tasks.

Sha [13] introduced WSSM-Q by using a QoS management module to evaluate the quality of Web services in compared to the user's requirements. In this work, they considered only four non functional attributes: availability, price, latency and performance. Their work consists of two levels: QoS information collecting and, QoS information processing. In the QoS information collecting stage, they use a monitor facility based on a handler provided by the JAX RPC and JAX WS specifications. The quality information of the requested Web service from the time a user sends a request until he receives the result is collected and sent to the management module. In the QoS information processing level, all the quality data are calculated and stored in a QoS repository.

Yan and Piao [14] modeled a vector-based algorithm by extending UDDI to cover all QoS information of advertised Web services. Providers need to add an entity (QoSInformation) to the advertisement that contains the information about all QoS attributes. Each QoS parameter is defined as a 5-tuple: <*attributeName*, *attributeType*, *attributeValue*, *attributeUnit*, *constraints*>, which represented the name, type, value, unit and related constraints of the parameter. To store the QoS information, they generated a tModel QoS information structure. To accomplish this goal, they devised an external file to minimize the modifications to UDDI registry. The external file might be maintained by the provider. In the ranking process they considered both parameters requested in the query and QoS information of a matched service as two vectors and calculated the distance between the vectors. Those services with lower distance value appear on top of the ranked list.

Liu and He [15] proposed a vector-based ranking algorithm to measure the goodness of the matched services and make recommendations to users based on their requirements. They modeled all the published services in a vector. They also modeled the query QoS requirements and their related weights in an n-dimension binary vector. The weights are required to be assigned by consumer. Then they calculated the distance between the query vector and the published. In their framework, all services are sorted based on their distance scores. The framework solves the problem by returning the final ranked list based on a minimum score.

Constraint Programming approach was introduced to specify attributes to all registered and also requested Web services. Degwekar et al. [16] extended WSDL to support specifications and constraints assigned to attributes. The extension helps the matching process to select correct candidates. To accomplish the matching process, they proposed a Constraint Satisfaction Processor (CSP). As an input, the processor takes the constraints of a requested service and also specifications advertised by a provider. In CSP, a normalization technique is used to normalize constraint specifications of all numerical attributes into "interval sets". The output determines which Web service and to what degree it matches with the user's request.

Li et al. [17] proposed a solution based on Hierarchical Constraint Logic Programming to specify the descriptions of the non-functional properties of all attributes. The model is composed of three stages: matching phase in which the tuple of request properties and offered solutions that need to be evaluated are defined. The second stage is the local evaluation phase in which the items of each tuple are compared based on their functional capabilities. Finally there is a third stage named global evaluation phase in which the matching degree between offered service and the requested item is evaluated by adopting a Simple Additive Weighting (SAW) scheme and a multi-criteria decision making technique.

Kritikos and Plexousakis [18] devised two CSP-based QoS-based WS discovery algorithms. The first simple algorithm covers only unary constraints. The second framework is based on Constraint Satisfaction Optimization (CSO) techniques and supports n-ary constraints. The WS discovery process in this model is composed of four steps: alignment, matchmaking, optimization, and selection. In the alignment step the QoS metrics of advertised QoS and user's demand are aligned by using a defined ontology (OWL-Q). In the matching step, the matched offers with the user's demands are returned. If the constraints of the demand are over-constrained, the matching step won't return any result. In this case the constraints of the demand are relaxed in the optimization level. In the last step, using a CSO the results are ordered based on the weights provided by user.

The model proposed by Li et al. [19] tries to provide a framework for describing the non-functional properties to be used in the service evaluation process. They used a Policy-Centered Meta model (PCM) to implement the framework. The evaluation is done by using a CSP in which the user's requirements are modeled in a hierarchical style. They used a satisfaction and an error function under each level to evaluate services. The role of the functions is to measure the similarity degree between the non-functional properties specified in a query and the non-functional properties of offered services. Then they used a multi-criteria decision making method to rank the Web services.

It is argued that Constraint Programming techniques are not able to solve problems including discrete variables. To address the drawback of CP models, MIP (Mixed Integer Programming) was introduced which supports linear constraints and both continuous and discrete variables. The framework is based on a set of variables and a set of constraints associated with the variables and a function to be minimized or maximized. Kritikos, and Plexousakis [20] used MIP solution in the matchmaking process. In their model, after the query is submitted, the specifications in the query are checked to see if they have right syntax or not. If they pass this stage, they will be transformed into a MIP solver engine. If there is any MIP solution applicable to the document, then it is stored in the MIP repository. At the end, a matchmaking algorithm is adopted to check all matched documents in the MIP repository with the requirements.

MCDM (Multiple Criteria Decision Making) or MCDA (Multiple Criteria Decision Analysis) is a family of methods being used in the decision making environment. Herssens et al. [21] presented a method based on the MCDM technique that provides the definition of a global priority constraint in the service discovery algorithm. In this model, a global priority constraint is defined to indicate the priority orders of QoS parameters. Then the global priority constraints are specified based on a MCDM technique (PROMETHEE) in which the pair-wise comparisons of items are executed by considering the deviation between the evaluations of the alternatives. The priority orders are converted into priority weights to be used in the selection process.

Tran et al. [22] introduced a model based on Analytic Hierarchy Process (AHP) which is another method to solve the MCDM problems. The model consists of three steps: first the complex problem is modeled in a hierarchy structure to present the relation between the overall goal, the criteria and the solutions. In the second step, the criteria items are compared to compute the priorities over the criteria and provide their relative ranking relation. In the last step, the ranks of all criteria are combined to indicate the ranking relation between all offered services.

## 2.3. Rank Aggregation Methods

Many Web service discovery and ranking methods are inspired by the effective techniques developed in the database community. Rank-based aggregation technique proposed by Aslam and Montague [23] is one of the methods used in this context. In this model, first the services are ranked in different lists based on each individual attribute, and then the algorithm combines different ranked lists to compute the final ranked list.

Rank aggregation problem is the problem of how to aggregate *m* ranked lists generated by *n* sources. According to the latest researches, there are two types of rank-based aggregation methods: supervised rank aggregation technique which relies on the training data and un-supervised rank aggregation method with no need of the training data. Unsupervised rank aggregation technique is categorized in two groups: positional methods and Majoritarian techniques. Positional methods generate the final ranked list based on the combination of all ranking scores gained by summing all the positional values of each element in each ranked list. The most common method in regards with the rank aggregation method is the Linear Score

Combination method in which the scores of items are aggregated by some operators such as weighted sum to compute the final ranked list.

Another important algorithm in this context is referred as Borda-Fuse proposed by Bartell et al. [24]. It is considered an effective algorithm to rank a set of data points. The algorithm was introduced to solve the voting problem. Suppose we have *n* voters to rank a fixed set of *m* data points. As a result we will receive *n* ranked lists including *m* data points. For each individual ranked list, the top ranked item receives *m* points, the second candidate receives $m - 1$ points, and so on. The last item in the list receives 1 point. The final score for each candidate is calculated based on the summation of all *n* points assigned to the candidate. The item with the most points will be ranked the best. It is a very simple procedure, which has been proved to be effective.

Another positional algorithm that can be named in this area is Median-Rank aggregation method introduced by Fagin et al. [25], in which the candidate documents are ranked based on their median ranks. Given *m* data points and *n* lists of values assigned to each point, the ranking process works as follows:  first, all *m* documents are sorted according to their values in the lists, thus *n* ranked lists will be returned; then, the final rank list is computed as the median of the positional values of each element in the rank lists. The final ranked list is generated by sorting the median value of each element. This method is not able to support ties.

Footrule Optimal Rank Aggregation method [26], [27] is another type of positional rank aggregation methods. The ranking prototype is used to minimize the Spearman Footrule Distance (SFD) from the input rankings. Based on this theory, for any given two rank lists $l_i$ and $l_j$ in a set of *n* lists, the SFD is computed as follows:

$$\text{SFD } (l_i, l_j) = \sum_{i=1}^{n} | r_i^{l_i} - r_i^{l_j} | \qquad (2.1)$$

Where $r_i$ is the ranking position of candidate *i* in each related list. The lower value of SFD shows the more similarity between the two mentioned lists.

Majoritarian rank aggregation approaches are another type of unsupervised rank aggregation methods. In this type of algorithms every item is compared with another candidate [28]. The method consists of repetitive steps. First they considered a list of all candidates and then each item in the list is compared with the next one. The winner stays in the list, but the loser will be removed from the list. The comparison steps are repeated until there is no other item in the list to be compared. This method suffers from low speed, as the number of comparisons gets larger, when the number of items in the dataset increases.

Condorcet-Fuse method proposed by Montague, and Aslam [29] is one of the voting models based on Majoritarian rank aggregation technique. This model is also based on pair-wise comparisons. Each candidate is compared to all other items in the dataset in terms of all QoS attributes. The model works based on a theory that the item which wins in majority of pair-wise comparisons, will appear in the final ranked list. However this method suffers from high computation time, and lack of capability to deal with ties, i.e. the situation that it is not feasible to choose a winner from a comparison contest.

## 2.4. Skyline Operation

There is another type of matchmaking and ranking algorithms based on Skyline query concept which is a dominant topic in the database field .The skyline operation was introduced by Kossman et al. [30] to solve maximum vector problems. The model calculates and filters the desired points relevant to a query and returns all possible solutions among a large set of data points in a given domain. Suppose that a client is seeking cheapest hotels closer to the shopping centers. It is difficult to choose a hotel between all possible options, as closer hotels to the shopping centre tend to be more expensive. According to the skyline operation, the desired hotels are those not worse than the others in both dimensions. The ultimate set of desired hotels is called as skyline points. Skyline points are composed of services that are not dominated by other services. A service $S_i$ dominates service $S_j$ if it is better than $S_j$ in at least one attribute and not worse than the service in other attributes. Skyline points assist consumers to select their desired service easier based on their preferences. From the graphical point of view, the name of skyline was selected for the computation of result set.

In context of the skyline query field, Papadias et al. [31] introduced a progressive algorithm which relies on Branch and Bound Skyline (BBS) based on a nearest-neighbor search method. On a given set of points, this model computes the skyline points based on their distances to a query point in an ascending order. In this work they first indexed the data by applying an R-tree technique to reduce the computation cost by decreasing the number of pair-wise comparisons. Then they computed the dominance relationship between each two services. They argued that in their framework any pre-computation functions would not be required. BBS is widely used in multi-criteria optimization problem.

To extend the Skyline query model to relational databases, [32], [33] presented a new algorithm called Sort Filter Skyline (SFS) model. They implemented their model based on a sorting technique. According to their theory all data points are sorted by using a sorting technique and considering a monotone function. In other words, SFS sorts all candidates that maximize the scoring function in an ascending order. After sorting the data, the services which dominate the other services over most attributes will appear in upper positions. Thus the number of pair-wise comparisons will decrease. Any service with the best score over the monotone function will appear in the skyline list. This method is used extensively and is a fundamental structure for methodologies invented later. This model is also considered as a baseline for comparison purpose in many works.

Inspired by skyline query concept, Han et al. [34] introduced a new Fast Item Skyline (FI-Skyline) algorithm for matching and ranking Web services. Their design is a solution for defining the appropriate Web services as skyline points. In their model they adopted Latice-based indexing method and R-tree sorting technique to index and sort data. The low cardinality attributes are transformed to dimensional lattice and then they use an indexing technique to set an R-tree and to index the data. To compute the skyline query, a topological sorting order is used to traverse the R-tree. They claim their work outperforms the state-of-the-art skyline method.

Skoutas et al. [35] proposed a new algorithm for ranking and clustering Web services according to the dominance concept. Their model supports multi criteria matching without combining matching scores of each individual parameter. The model combines top-k queries and skyline operation. A threshold is also considered in this work to compute the probability of being in skyline for each service. The algorithm is composed of 3 steps: 1- Select the services that their probability to be in the skyline above the threshold, 2- Select k representatives from the list returned in the previous step, 3- Form the clusters by assigning the other services to their related cluster. Their model also reflects the trade- offs between matched parameters.

## 2.5. Chapter Summary

In this chapter, we reviewed some of the popular Web service discovery and ranking algorithms from semantic and syntactic based approaches. In the semantic-based category, ontology definitions play vital roles in measuring the similarity between requested and published

Web services. All the queries and advertised properties are transformed to a semantic-based model, and then the discovery and ranking procedures are applied. There are two major issues related to this category: 1- The time to process the request is long. 2- There is no standard ontology definition for different problems. On the contrary the syntactic-based algorithms are introduced which rely on the syntax data, and usually are faster. We also reviewed some other approaches such as Skyline operation and Rank aggregation techniques that were originally introduced in context of different fields such as Web search or database context.

Most of the reviewed models are reasonable work; however they suffer from some deficiencies: 1- Many of them are complicated methods based on data indexing and sorting techniques which generally have a longer processing time. 2- They mostly either ignore the role of users' requirement or their methods require users to compute the importance degree of each parameter. On the other hand they put more load on users. Consumers tend to use applications which run fast without involving them in the computations. 3- They only considered a limited number and a few types of attributes (mostly numeric type), while in reality there are various types of data.

In our work, we tried to address the above mentioned issues by developing a simple, but effective method which considers user requirement as an important factor in the ranking process without imposing any further load on consumers.  In our model, we also take into account different number and types of data.

# CHAPTER 3

# WEB SERVICE DISCOVERY AND RANKING FRAMEWORK

## 3.1. Overview

Web service selection and discovery system is essential to provide clients with proper results according to their requirements. It is impossible to fulfill this task without considering the ranking relation between thousands of available candidates with similar functionalities. Ranking process is a fundamental step in a Web service selection system, as it integrates the results gathered from previous stages (functional and non-functional matching process) and presents them to the requestors. In this work we focus on the ranking process by considering user's QoS requirements. This chapter begins with an overview on functional and QoS requirements, and then we review one of the Skyline algorithms in details. We use this algorithm as a baseline for our comparison purpose. The main reason is that Skyline is well-accepted algorithm in database area for the multi-criteria based selection problem, and recently is being used in Web service selection area due to the accuracy of the generated results. However, efficiency is one of the big concerns on this algorithm. We would like to see whether a much simpler algorithm with higher efficiency could achieve the similar level of accuracy. The simple algorithms we chose in this thesis are a vector-based (Distance-based) algorithm and a rank aggregation algorithm (Borda Fuse). Since both of them (also the skyline algorithm) do not consider the actual user requirements at all. We introduce the enhanced algorithms by taking into account the user's QoS requirements. The goal of the proposed models is to provide a simple and effective method for generating a ranked list of desired Web services with consideration of user's requirements. The

QoS data considered in this research include different types of data such as interval, list, Boolean, numeric, etc.

## 3.2. Functional and QoS Requirements

Requestor is an important entity in the Web service discovery and selection model. The discovery process starts from the time when a user submits a request for a desired service with a specific functional task such as "travel booking" or "weather report". There are thousands of services to be retrieved in respond to the query. More and more services with the same functionality are coming to the market. Functional property of a service is not sufficient to make the service qualified for different users. The best way to narrow down the results for a user is to classify and rank the items based on their QoS properties. The QoS property of a service indicates how a service behaves. The QoS attributes are provided by either service providers, or monitored by some agents. An example for a QoS attribute is "Availability" which is a service quality to represent the probability of a service being up and running without a breakdown, when a user sends an invocation request. "Cost" is another quality attribute indicating the fee that a user needs to pay for using a particular service. There are different QoS features to represent different aspects of a service. In this thesis we concentrate on QoS properties as the deciding factors for our selection framework.

### 3.2.1. QoS Attributes Types

QoS attributes hold two different types of values: a single value such as numeric, Boolean, string, etc.; or multiple values such as list, set, range or fuzzy type. Most attributes considered in majority of the current works are numeric types such as response time: = 0.10 ms or availability: =90 %. However it is not accurate to represent all types of QoS value in this way, as different invocations may have different values. They might need to be represented as an interval such as (0, 0.10).

There are also other QoS parameters with different types rather than a single numeric value. In most cases it is infeasible to represent them in numeric format. As an example attribute "supported standards" is mostly defined as an unordered list such as: ["WSDL1.1", "WSDL2.0", "SOAP1.1", "SOAP2.0", "UDDI3.0", "UDDI3.4"]. This attribute is defined to present which standards are supported by a Web service.

To the best of our knowledge multiple-values type of attributes was not considered in the majority of current approaches. The models in which the multiple-values type were taken into account, did not study the impact of these data types on the efficiency of algorithms using large sized datasets. In most of the existing models, the QoS values are considered as a single number. In this research we consider various attributes with different types including numeric, Boolean, data interval, and unordered list.

### 3.2.2. QoS Tendency

In order to evaluate two attributes fairly, we need to consider their direction or tendency of their values. In other word, if the tendency of the attribute is positive, it means a bigger value is better. On the contrary if the tendency is referred as negative, it means smaller values are preferred. For example for attribute "cost" the smaller value is usually preferred, so the tendency of this parameter is negative, whereas for attribute "availability" the bigger value indicates a better quality for the specified parameter, so the tendency is positive.

### 3.2.3. QoS Data Comparison Rules

There are different rules required to be considered in ranking two different Web services in terms of their QoS values. To denote the relative rank between two services $S_i$ and $S_j$, if $S_i$ is better than $S_j$ over quality *q*, we represent it as $S_i^q > S_j^q$. In this research, we only consider 4 data types including Boolean, numeric, list and range. For each value type we use the following rules to evaluate the relative ranking between two services:

Boolean: in case of an attribute with a Boolean type, we have two different scenarios. In case a "True" value means a better attribute, the service with "False" value is the worse service. On the contrary if "False" value means a better attribute, then the service with "True" value is the worse.

Numeric: When a QoS property has a numeric value, the evaluation depends on the tendency of the attribute. If the tendency of Qos attribute is positive, the service with bigger value is ranked higher, whereas if the tendency of the attribute is negative, the service with smaller value is considered a better service.

List: In case of comparing an attribute with unordered set type, in this research we evaluate services according to the number of their common items with the user request. The service which has a larger number of similar items will be ranked higher.

Range: In terms of data interval values, we use the theory introduced by Sengupta, and Pal [36] to express the order relation between two data intervals and find the maximum data range. Suppose an interval data A= [$a_l$,$a_u$] where $a_l$ and $a_u$ are lower and upper bounds of the interval. According to their assumption, the range data might also be represented as (*m*(A),*w*(a)) where *m*(A) and *w*(A) are mid-point and half-mid of the interval A. *m*(A) and *w*(A) are computed as follows :

$$m\text{ (A)}=1/2(a_l+a_u), \qquad (3.1)$$

$w$ (A)=1/2($a_u$-$a_l$) (3.2)

Suppose A= [$a_l$,$a_u$] and B= F [$b_l$,$b_u$] as two intervals in real line$\mathbb{R}$ . Based on their theory a function $f$ is used to indicate the grade of acceptability that first interval is larger than the second interval. The function is defined as:

$$f\,(\mathrm{A, B}) = \frac{m(B) - m(A)}{w(B) + w(A)} \qquad (3.3)$$

where $w$ (A) +$w$ (B) $\neq$0.

According to the theory, the function $f$ is interpreted as follows:

$$f\,(\mathrm{A, B}) \begin{cases} =0 & \text{if } m(\mathrm{A}) = m(\mathrm{B}), \\ >0, <1 & \text{if } m(\mathrm{A})<m(\mathrm{B}) \text{ and } a_u > b_l\,, \\ >=1 & \text{if } m(\mathrm{A})<m(\mathrm{B}) \text{ and } a_u <= b_l \end{cases} \qquad (3.4)$$

If $f$ (A, B) =0 then the assumption that the interval A is better than interval B fails. In that case, the ranking order is based on the value of $m$ (A) and $m$ (B). If $f$ (A, B)>=1, it means that the first interval A is larger than the second interval B. If 0<$f$ (A, B) <1, then the function $f$ interprets the degree of the acceptability of interval A to be smaller than interval B.

### 3.2.4. QoS Data Normalization

As QoS attributes have different value spans, and are measured in different units, it is hard to compare them accurately. In order to have a fair evaluation, all data need to be normalized into a common range for example [0, 1] to guarantee they are evaluated in the same span. For each query attribute ($q_i$), we take the maximum value as $q_{max}$ and the minimum value as $q_{min}$. Based on the direction of the attribute, the normalized value ($q_i'$) is calculated according to the following equations [37]:

1) If the tendency of attribute is negative:

$$q_i' = \frac{(q_{max} - q_i)}{(q_{max} - q_{min})} \quad \text{where } q_{max} \neq q_{min} \tag{3.5}$$

2) If the tendency of attribute is positive:

$$q_i' = \frac{(q_i - q_{min})}{(q_{max} - q_{min})} \quad \text{where } q_{max} \neq q_{min} \tag{3.6}$$

$$\begin{cases} q_i' = 1 & \text{if} \quad q_{max} = q_{min} \neq 0 \\ q_i' = 0 & \text{if} \quad q_{max} = q_{min} = 0 \end{cases} \tag{3.7}$$

The methods of normalizing interval data are mostly based on interval arithmetic. Let $q_i = [q_i^l, q_i^u]$ where $q_i^l$ and $q_i^u$ are lower bound and upper bound of the data range respectively. To normalize $q_i$ we use the following equation [38]:

$$q_i' = \left[\frac{q_i^l}{\sum_{i=1}^{n} q_i^u}, \frac{q_i^u}{\sum_{i=1}^{n} q_i^l}\right] = [q_i'^l, q_i'^u] \qquad i = 1, 2, \ldots, n \tag{3.8}$$

where

$$q_i'^l = \frac{q_i^l}{\sum_{i=1}^{n} q_i^u} \quad \text{and} \quad q_i'^u = \frac{q_i^u}{\sum_{i=1}^{n} q_i^l}$$

In order to normalize the attributes of an unordered list type such as "Supported Standards" attribute, we assign a numeric value to each QoS parameter based on the number of items in the assigned list. Then we adopt one of the equation (3.5), (3.6) or (3.7) for the normalization process.

Boolean types are considered as numeric type. We consider 1 for True values and 0 for false values. As a result, the normalized value will be either 1 or 0 based on the value of the QoS property.

## 3.3. Skyline Operation

Skyline operator in the context of database was first introduced by Kossmann et al. [30] as multi-objective optimization concept to solve maximum vector problems. On a given set of data, skyline consists of a subset of data points that are not dominated by other entities. According to their definition, in a given domain, a point dominates another one, if it is as good as or better than the other point in terms of all requirements, or better in at least one requirement. Real Estate industry is a sample field in that we can use Skyline operation to assist the clients to select the best options according to their requirements. As an example we consider the price of an apartment and its distance to the public transportation facility (PTF) as two dimensions in a small dataset of rental apartments as in Table 3.1. In this example, we ignore the other properties of the apartments and only focus on the two mentioned attributes. In this case the ideal choice would be the apartments with the lowest price and the minimum distance to PTF. However there is no apartment in this set of data that is better than the others on both criterions. This is the fact that we experience in real life too.

Table 3.1- Example of a list of rental apartments

| Apartment | Price ($) | Distance to public transportation (mile) |
|---|---|---|
| A | 1050 | 0.9 |
| B | 1200 | 0.8 |
| C | 1220 | 0.78 |
| D | 1100 | 0.85 |
| E | 1420 | 0.95 |
| F | 1250 | 0.78 |
| G | 1300 | 0.7 |
| H | 1350 | 0.65 |
| I | 1370 | 0.91 |
| J | 1375 | 0.7 |
| K | 1400 | 0.7 |
| L | 1200 | 0.9 |
| M | 1250 | 0.87 |
| N | 1150 | 0.95 |
| O | 1400 | 0.99 |

To refine the results and provide a more desirable list based on user's demand, we remove all those options that are worst on both criterions. In our case, apartments E, F, I, K, J, L, M, N and O may be eliminated as there are still some better options in the list than these items. This will result in options on skyline represented in Table 3.2.

Table 3.2- Results in the skyline

| Apartment | Price (Dollar) | Distance to PTF (Mile) |
|---|---|---|
| A | 1050 | 0.9 |
| B | 1200 | 0.8 |
| C | 1220 | 0.78 |
| D | 1100 | 0.85 |
| G | 1300 | 0.7 |
| H | 1350 | 0.65 |

Using a skyline model, we can filter out a set of best options according to the required criteria. To refine the skyline list according to a user's demand, we can execute any SQL command on the filtered dataset. Suppose a client is looking for an apartment with the price between $1000 and $1300 with its distance to the public transportation facility no more than 0.9 miles. Based on the available data in the dataset, the dashed line in Figure 3.1 shows the skyline of the results in terms of the requirements.

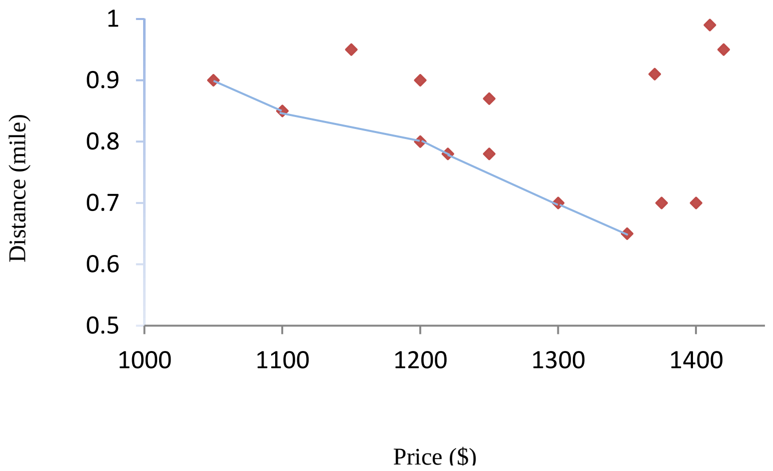

Figure 3.1- Skyline results based on the user's requirements

The solid line is referred as the skyline due to its graphical representation. Any points above the line do not meet either one or both requirements of a query. To generate the skyline, any monotone scoring function can be considered. Those candidates that maximize the scoring function will appear in the skyline. There are different ways to compute the skyline. SFS algorithm [33] was proposed as a prominent skyline operation method based on sorting data to reduce the computation time. In this framework a given dataset is sorted based on an aggregation function over the constraints in the query. The result will be saved in a table and then all sorted items are compared to each other to locate the points that dominate the other items. At the end, all services that are not dominated by the rest of services are saved in an output table. As it is evident, user's requirements have not been considered in the model. However after generating the output table, we can execute different queries on the result table based on user's demand. All services which meet the requirements will appear in the final list in a descending order. Figure 3.2 represents the steps of the SFS algorithm for computing skyline. The final list contains all optimal services sorted based on their dominance scores.

```
Input:   A set of services $T$
Dimensions (number of attributes) $d$
Output:  Ranked list of Web services $S$
1    begin
2      initialize $T'=\emptyset$;
3      sort all data of $T$ in descending order and save them in $T'$;
4      initialize $S=\emptyset$;
5      move the first record of $T'$ into $S$;
6      while $T'$ is not empty
7          for each record $t$ in $T'$:
8              compare $t$ with all records in $S$:
9              if $t$ is dominated by some records in $S$ then
10                 remove t from $T'$;
11             else
12                 move t from $T'$ to $S$;
13     returns $S$;
14   end;
```

Figure 3.2-Sort-Filter-Skyline algorithm

## 3.4. Enhanced Borda Fuse Algorithm

In this section we explain the original Borda Fuse algorithm in details and then provide the extended method by considering user's requirements through an example.

### 3.4.1. Basic Borda Fuse Algorithm

There is another category of methods proposed for ranking data items which is called rank aggregation methods. One of the efficient methods in this context is Borda Fuse [24]. This model was proposed to solve the voting problems in different areas. In the context of Web service discovery, a service which appears in the highest positions in the most ranking lists will receive higher ranking score. In this model all services are first ranked in different lists in terms of different QoS attributes. Each service is assigned a score based on its positional value in each individual ranked list. Then the final ranked list is generated by computing the summation of all obtained scores from all ranked lists. The required steps of the algorithm are presented in Figure 3.3.

```
Input:   A set of m services T
         Dimensions (number of QoS attributes) k
Output:  Ranked list of Web services S
1     begin
2         sort T based on each dimension to get k ranked lists;
4         initialize S=∅ ;
5         for each service s in T
6             initialize score=0;
7             for each individual ranked list
8                 assign score= score + ((m- positional value of s in the list) +1);
9                 move s and score to S;
10        sort S based on score;
11        returns S;
12    end;
```

Figure 3.3- Borda Fuse algorithm

### 3.4.2. Enhanced Borda Fuse Algorithm with Consideration of User's Requirements

The Borda Fuse algorithm suffers from an important deficiency that may affect the accuracy of the results. In this model the user's actual request is ignored which is an important factor in selection systems. For different requirements from different users, the output of the algorithm is always the same. To overcome this issue, we improved the algorithm to cover user's requirements in the ranking process. To be more specific we include the requested constraints to the rank lists before calculating the scores.

Suppose we have *n* services retrieved from the matching phase of the Web service discovery, and *k* required QoS attributes determined by user. Based on the original method, *k* ranked lists will be generated according to each attribute. For each service, the ranking score is calculated by summing up the positional values of the service in each ranked list. As an example, suppose we have 5 similar functional Web services with different quality attributes as represented in Table 3.3. There are 4 criteria required to be considered for ranking these services.

Table 3.3- Similar Web services with different QoS values

| Criteria | Availability % | Response time(ms) | Cost($) | Reliability % |
|---|---|---|---|---|
| Tendency | High | Low | Low | High |
| Required QoS values | >=92 | <=4 | <=3 | >=85 |
| Service1(S1) | 93 | 1.8 | 2.5 | 80 |
| Service2(S2) | 94.5 | 2.65 | 2.3 | 92 |
| Service3(S3) | 92.5 | 2.1 | 2.6 | 85 |
| Service4(S4) | 96 | 2.0 | 3.0 | 90 |
| Service5(S5) | 94 | 1.95 | 3.2 | 84 |

According to the original Borda Fuse algorithm all services are first ranked separately based on each constraint. In this case we have four ranked lists of services based on each criterion and tendency of the attribute. The generated ranked lists are illustrated in Table 3.4.

Table 3.4- Ranked lists based on 4 QoS attributes

| Ranked list based on availability | Ranked list based on response time | Ranked list based on cost | Ranked list based on reliability | Final Ranked list |
|---|---|---|---|---|
| S4 | S1 | S2 | S2 | S2 |
| S2 | S5 | S1 | S4 | S4 |
| S5 | S4 | S3 | S3 | S1 |
| S1 | S3 | S4 | S5 | S5 |
| S3 | S2 | S5 | S1 | S3 |

The final ranking score is calculated by adding the positional value of each service in each individual ranked list. The problem is that, the actual user request plays no role in the ranking score. With different requirements, we will receive the same result. To involve user's requirements in the algorithm, we consider query attributes as a sample service Sq and add it to the list of offered services. As a result we will have new ranked lists including Sq as indicated in Table 3.5. We can notice that the position order of service S3 and S5 has changed in the new ranked list. Service S5 does not meet the requirements for the last 2 attributes.

Table 3.5-Ranked lists based on the user's requirements

| Ranked list based on availability | Ranked list based on response time | Ranked list based on cost | Ranked list based on reliability | Final Ranked list |
|---|---|---|---|---|
| S4 | S1 | S2 | S2 | S2 |
| S2 | S5 | S1 | S4 | S4 |
| S5 | S4 | S3 | S3 | S1 |
| S1 | S3 | S4 | **Sq** | S3 |
| S3 | S2 | **Sq** | S5 | S5 |
| **Sq** | **Sq** | S5 | S1 | **Sq** |

The scores assigned to services depend on their position in each ranked list. We assign negative score to those services which appear in the each rank list after Sq. The negative score for each service is computed according to position of service in the new ranked list. Suppose *n* as the number of services, $S_i$*.position* as the position value of service $S_i$ in each ranked list, S*q.position* as the positional value of Sq in the new list with considering user's requirements. Considering the new list, if a service is in a higher position than *Sq*, then the score for each list is computed as follows:

$$Rankscore = (n - S_i.position) + 1$$

In case a service has a lower position than *Sq* in the ranked list, the score is calculated as:

$$Rankscore = \text{Sq.position} - S_i.position$$

The total score ($Score_{Si}$) for each service is calculated as:

$$Score_{Si} = \sum_{i=1}^{m} w_i \; Rankscore$$ where *m* is the number of QoS attributes.

The final ranked list could be generated by sorting the final scores in a descending order.

## 3.5. Enhanced Distance-Based Algorithm

In this section we discuss the classical Distance-based algorithm in details and then provide the improvement algorithm with consideration of user's requirement. In the improved model, we not only consider the optimal value of each requirement, but also take the real consumer's demand into account.

### 3.5.1. Basic Distance-Based Algorithm

In this model, the problem of matching and ranking Web services based on the requirements is considered as a distance measurement problem [14]. To solve the problem, a published Web service is modeled in a vector including its *n* QoS attributes. The optimal values of each QoS attributes are also modeled in another vector. The distance between the two vectors is measured according to Weighted Euclidean Distance formula. Then in the ranking process, the ranked list is generated by sorting the distance values in an ascending order. The required steps of the algorithm are indicated in Figure 3.4.

```
Input:   A set of Web services including $n$ QoS attributes
         A vector $W_n$ to represent $n$ weights for $n$ QoS attributes
Output:  Ranked list of Web services: $S$
1   begin
2       initialize $S=\emptyset$;
3       for each service $s$
4           arrange $n$ QoS attributes of service $s$ in a vector $S_n$;
5           arrange optimized values of QoS attributes in a vector $Q_n$;
6           normalize items of vector $S_n$ to generate vector $S_n'$
7           normalize items of $Q_n$ to generate vector $Q_n'$;
8           calculate the distance score between $S_n'$ and $Q_n'$;
9           move distance score to S;
10      generate ranked list $S$ by sorting distance scores;
11      returns $S$;
12  end;
```

Figure 3.4- Distance-based algorithm

The Distance-Based algorithm is an efficient method to generate a list of ranked services. However, the original algorithm only considers optimal value for each attribute. The algorithm can be improved to cover user's requirements to be more realistic.

### 3.5.2. A Motivating Example

Suppose there are 3 similar Web services with 4 QoS attributes including "availability", "reliability", "composability" and "response time" offered by service providers as indicated in Table 3.6. The results were computed based on a query (availability >=92%, reliability>=80%, composability>=85% and response time<=2.5ms).

Table 3.6- Values of 3 similar Web services with 4 QoS attributes

| Criteria | Availability % | Reliability % | Composability % | Response time ms |
|---|---|---|---|---|
| Tendency | High | High | High | Low |
| Required QoS values | >=92 | >=80 | >=85 | <=2.5 |
| Service1( $S_1$ ) | 94.5 | 90 | 92 | 2.3 |
| Service2( $S_2$ ) | 93 | 99 | 89 | **3.5** |
| Service3( $S_3$ ) | 92.5 | 90 | 91 | 2.4 |

By applying the original Distance-based algorithm to the sample dataset, the ranked list is retrieved as: $S_1 > S_2 > S_3$. Service $S_2$ appears before service $S_3$ in this list, however $S_2$ does not satisfy the fourth criteria specified in the query (response time<= 2.5ms). Although Service $S_3$ meets all the requirements, it appears on the bottom of the list after $S_2$, as it does not acquire sufficient similarity score in compared to the optimal value of the second attribute (99%). The large difference between the values of each service for attribute (reliability) has led $S_2$ to obtain a higher ranking position in the list. If we consider the user's requirement in the model and apply the improved method, the result list based on the same data is returned as follows: $S_1 > S_3 > S_2$. The service $S_2$ is still in the ranked list, but appears after those services that satisfy all of user's requirements and closer to the optimal values. In fact by penalizing this service for not satisfying all requirements, we reassure that the list presented to the consumer could better meet his actual demand.

### 3.5.3. Improved Distance-Based Algorithm with Consideration of User's Requirements

We model the values of n QoS attributes of a service S as a vector: $Q_s = (Q_{s1}, Q_{s2}, ..., Q_{sn})$ and consider the values of QoS requirements requested by a consumer as a vector $Q_r = (q_{r1}, q_{r2}, ..., q_{rn})$. We also set the consumer's preferences values on each QoS attribute in a vector $P_r = (p_{r1}, p_{r2}, ..., p_{rn})$ where $p_{ri} \in [1, n]$. User needs to assign a number between 1 and *n* as the importance degree for each constraint. If the consumer has no preferences for an attribute, *n* will be considered as the preference value for that specific parameter. We represent the vector of weights assigned to attributes as: $W = (w_1, w_2, ... w_n)$, where $\sum_{i=1}^{n} w_i = 1$, and $w_i \in (0,1)$. We set $p_{rmax}$ as the maximum value in vector $p_r$ and use the following equations to compute the weight for each attribute:

$$w_i = \frac{p'_{ri}}{\sum_{i=1}^{n} p'_{ri}} \qquad i=1,2,\dots n \tag{3.9}$$

where $$p'_{ri} = (p_{rmax} + 1) - p_{ri} \tag{3.10}$$

If we consider $Q_o = (q_{o1}, q_{o2}, ..., q_{on})$ as the vector of the optimal values of the QoS attributes, then the score assigned to service S ($Score_{os}$) is computed based on its distance to the vector $Q_o$. The distance score is calculated based on the Euclidean Distance formula. Up to this point, only optimal values of attributes were considered in the algorithm. In order to apply the real query values in the model, we also calculate the distance between the QoS of service S and the real requested constraints specified in the query. The final distance score (*dis* (S)) which indicates the distance between a service and both optimized and required QoS values is calculated as explained in the following paragraph:

In order to evaluate the ranking relation between the services, we need to calculate the distance between each QoS attribute of each service and the optimal OoS value. The optimal value is the best value on a QoS property and it depends on the tendency of the attribute. Based on the Euclidean distance measurement theory, the following formula is used to calculate the score for each service $S_i$ based on its distance to vector of optimal OoS( $Q_o$) :

$$Score_{osi} = | Q_{si} - Q_o | = \sqrt{w_j (q'_j - q'_{oj})^2} \qquad (3.11)$$

Depending on the tendency of the QoS parameters, the normalized values for numeric, list and Boolean attributes, are computed based on equation (3.5), (3.6), or (3.7). For any attribute of the range type, the normalized value is calculated based on equation (3.8).

In order to calculate the distance between each service $S_i$ and the requested QoS ($Q_r$), we penalize the services that fail to meet any requirement requested in the query. To fulfill this, we consider a vector: $P_i = \{p_{i1}, p_{i2}, ..., p_{in}\}$ where $p_{ij} \in [1 \ldots n]$). The vector $P_i$ includes 0 or 1 based on whether the service $S_i$ meets each requirement in the query or not.

We obtain the distance between each service and the real required QoS ($Q_r$) as follows:

$$Dis_{si} = | Q_{si} - Q_r | = \sqrt{p_{ij} w_j (q'_j - q'_{rj})^2} \qquad (3.12)$$

In equations (3.11) and (3.12), $q'_j$ refers to the normalized value of the *jth* quality attribute ($q_j$), $q'_{oj}$ refers to the normalized value of the optimal value of the *jth* quality attribute ($q_j$), and $q'_{rj}$ is the normalized value of the *jth* requirement specified in the query.

In order to calculate the distance between two vectors of QoS and offered services for the attributes with interval data type we consider the required QoS vector: $Q_r = ([q_{1l}, q_{1u}],[q_{2l}, q_{2u}],...,[q_{nl}, q_{nu}])$ where $q_{jl}$ and $q_{ju}$ refer to the lower bound and upper bound of the *jth* QoS attribute. To compute the distance between vector $Q_r$, and the vector of QoS values of service $S_i$: $Q_{si} = ([S_{1l}, S_{1u}],[S_{2l}, S_{2u}],...,[S_{nl}, S_{nu}])$ where $S_{jl}$ and $S_{ju}$ are the lower bound and upper bound of the *jth* attribute of the service, we use the following equation to calculate the distance between service and the QoS constraints:

$$Dis_{si} = | Q_{si} - Q_r | = \sqrt{\sum_{j=1}^{n} p_{ij} w_j ((q'_{jl} - S'_{jl})^2 + (q'_{ju} - S'_{ju})^2)} \qquad (3.13)$$

In equation (3.13), $q'_{jl}$ and $q'_{ju}$ are the normalized values of lower bound and upper bound of the *jth* QoS attribute respectively. In this equation, $S'_{jl}$ and $S'_{ju}$ refer to the normalized values of the lower bound and upper bound of the *jth* attribute of the service respectively.

We assign $W_1$ as the weight for the score calculated by considering the user's requirements and, $W_2$ as the importance degree for the score calculated by considering only optimal values. Using Linear combination method, we compute the final score for each service as in the following equation:

$$Score_{si} = W_1 Dis_{si} + W_2 Score_{osi} \qquad (3.14)$$

In this research we consider equal weights for each factor as $W_1$ =0.5 and $W_2$ =0.5. The final ranked list may be retrieved by sorting all the services based on the final score in an ascending order.

## 3.6. Chapter Summary

In this chapter, first we described a promising algorithm in the context of discovery and ranking web services which will be used as one of the baseline algorithms for the comparison purpose. We also described two efficient algorithms, i.e. Distance-based and Borda Fuse from two different categories: vector-based and positional-based techniques. They will also be used as baseline models for our comparison purpose. Then we proposed our framework by improving the Distance-based algorithm. In the first step of our framework the distance-based algorithm is adopted to compute the distance between the optimal values of QoS parameters, and the values for services published by providers. Then we compute the distance between each retrieved Web service and the query over the attributes specified in the query. In our model, the final ranked list will be generated based on the overall scores gained from both phases. Finally we considered actual user's queries in the Borda Fuse approach by applying penalty scores for those services that do not satisfy user's demand. We consider the query as a Web service which is required to be considered in the ranking evaluation process. Those Web services placed after query in the ranked lists are the ones do not meet the requirements. Then they will be assigned with some negative scores based on their positional values in the lists. The selected services in the final lists are closer to user's demand.

# CHAPTER 4
# IMPLEMENTATION AND EVALUATION

In this chapter, we use various datasets with different sizes to study the performance of our improved algorithms and compare their performance with those of other algorithms such as the original Borda Fuse, Skyline, and the original Distance-based approaches. We consider the QoS parameters described by Al-Masri, and Mahmoud [39] as follows: reliability (%) - The ability of a service to perform the required tasks under specific conditions during a specified period of time, availability (%) – The probability the service is up and is available to be used, throughput (number of invocations/millisecond) – Total number of invocations managed in a period of time , successability (%) – Number of successful responses/ number of requests, response time (millisecond) – Required time to respond to a query, compliance (%) – The extent that a service complies with the WSDL specification, best practices (%) – The extent a service follows the standards defined in WS-I Basic Profile, latency (millisecond) – The measure of time delay to respond to a request, documentation (%) – Measure of documentation in the WSDL file and supported standards (list) – The standards supported by a Web service.

## 4.1. Implementation and Testing Environment

The algorithm has been implemented as a console-based application, using VB.Net language, in Microsoft Visual Studio 2008 environment with .Net framework 3.5, to be used in applications of selecting and ranking Web services. The experiments were run on a machine with Intel R Core (TM) CPU 2.30 GHz, 4 GB RAM, and installed Microsoft Windows 7 as the operating system.

## 4.2. Experiment Design

In all of the following experiments we use different subsets derived from the QWS dataset provided by Al-Masri, and Mahmoud [40]. The original dataset includes information of over 2000 web services available on the Web. The dataset includes real data for various QoS attributes such as response time, availability, throughput, successability, reliability, compliance, best practices, latency and documentation. The service name and its WSDL address are also included in the dataset.

We consider different scenarios for the experiments as follows:

1) Test the efficiency of our algorithm by using real datasets;

2) Compare the performance of our improved algorithms with three baseline algorithms using different datasets:

   2.1) Datasets with different sizes (increasing the number of the Web services);

   2.2) Datasets with different number of attributes (increasing the number of QoS attributes);

   2.3) Datasets with different types of attributes (such as single number, list and range);

3) Study the quality of the results of the improved algorithms

## 4.3. Evaluation of the Results

In the following sections we provide the results generated from different experiments to evaluate our improved algorithms in terms of two factors: (1) Execution time which represents the processing time from when the query is submitted, to the moment when the ranked results are returned and presented to the user. (2) The quality of the results which is evaluated based on the similarity degree between the ranked results obtained from the improved algorithms and that of the original models. We consider Skyline algorithm as the baseline for comparison with our improved algorithms in terms of efficiency. In the following descriptions, we use these abbreviations for each algorithm: DS for the original Distance-based algorithm, DS_I for the improved Distance-based algorithm, BF for the original Borda Fuse, BF_Q for the Borda Fuse algorithm with consideration of user's query, and SFS for the Sort Filter Skyline algorithm.

### 4.3.1. Evaluation on Efficiency

In this section we provide the results of different experiments conducted on different sized datasets and then analyze and compared the algorithms in terms of the processing time.

#### Experiment-1: Different Datasets with Different Sizes

In the following scenario we study the effect of increasing the size of dataset on the performance of each model. To fulfill the task, we run the applications on different datasets containing 10, 50, 100, 150, 200, 300, 500, 1000, 1500 and 2000 Web services. In this experiment we consider nine QoS numeric attributes including: response time, availability, throughput, successability, reliability, compliance, best practices, latency and documentation. We measured the execution time of algorithms by running each application 500 times and get the average value of the results. A sample query could be: response time<1000ms, availability>95%, throughput >20, successability>95%, reliability>80%, compliance>85%, best practice>80%, latency<50ms and documentation>50%. Then the query vector is set as : $q_i$=(1000,95,20,95,80,85,80,50,50). A sample preference vector for this query could be: $p_i$=(1,1,3,2,3,3,2,2,3).

The result of running each algorithm is presented in Table 4.1. As illustrated in Figure 4.1 the execution time of SFS algorithm is less than the others when the number of Web services is less than 150. When increasing the number of services, we observe that the performance of SFS is largely affected, whereas DS and DS_I algorithms run faster than the others. The average execution time of our improved method (DS_I) is a little more than the processing time of the original Distance-based for all different number of services, however the difference is very small. It is reasonable because DS_I takes extra steps of comparing service QoS values with the requested QoS values. When the number of Web services is increased, the execution time of DS_I still remains close to the processing time of the original DS algorithm, but the difference between its processing time and that of SFS is increasing. BF has higher performance than BF_Q, but the difference is small. The reason is that BF_Q requires one more step to add the query in the ranked lists. Due to the extra initial step in BF and BF_Q algorithms for generating different ranking lists based on different attributes, they process the query slower than DS and DS_I algorithms. BF and BF_Q outperform SFS on large sized datasets. When increasing the size of the dataset, the number of pair-wise comparisons in SFS algorithm is growing which will affect the processing time.

Table 4.1- Execution time of algorithms on different datasets

containing 9 QoS attributes

| | **Average execution Time (ms)** | | | | |
|---|---|---|---|---|---|
| **Number of Services** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| 10 | 6 | 7 | 9 | 12 | 5 |
| 50 | 22 | 22 | 27 | 31 | 16 |
| 100 | 40 | 41 | 45 | 50 | 37 |
| 150 | 60 | 61 | 67 | 74 | 69 |
| 200 | 79 | 81 | 84 | 90 | 114 |
| 300 | 105 | 107 | 120 | 129 | 148 |
| 500 | 199 | 201 | 214 | 224 | 301 |
| 1000 | 403 | 410 | 420 | 431 | 802 |
| 1500 | 643 | 646 | 706 | 723 | 1015 |
| 2000 | 873 | 887 | 1005 | 1029 | 1215 |

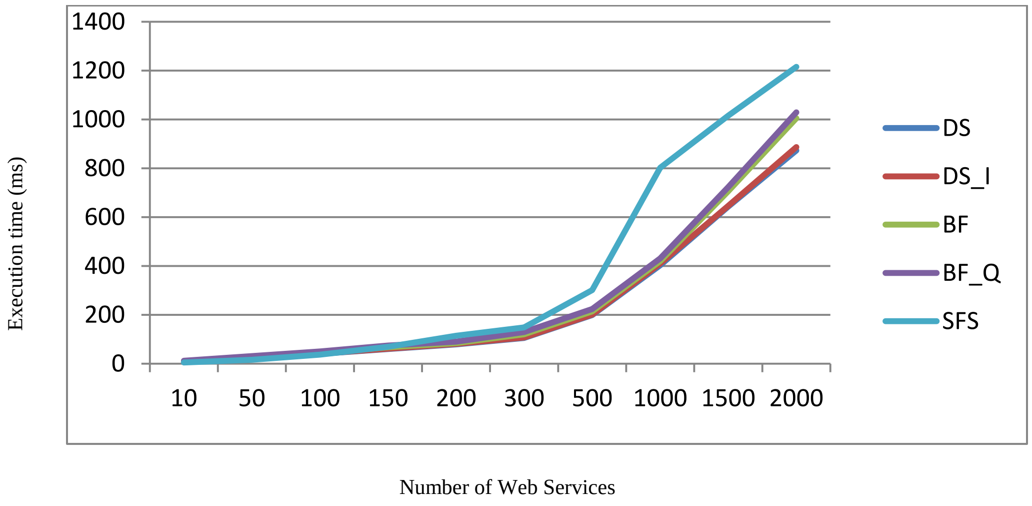

Figure 4.1- Comparison of execution time of 5 algorithms on different sized datasets containing 9 QoS attributes

**Experiment-2: Different Datasets with Different Number of Attributes**

To study the impact of increasing the number of QoS attributes on execution time of different algorithms, we did a different set of experiments with a combination of different number of Web services and different number of QoS attributes. We run the different algorithms on datasets with different sizes containing 10, 50, 100,150,200 and 1000 candidates. In this section, we present the performance results on each dataset in a separate table. The first row of each table shows the result when we have only one numeric attribute, i.e. response time (ms), the query is considered as (response time<=1000ms) and user's preference is set to 1. The second row shows the result for each algorithm when we have four QoS attributes: response time, availability, throughput and successability. The user's preference vector is set as (2, 1, 1, 3) and the query is submitted as (response time<=1000ms, availability>= 95%, throughput>=20, successability>=95%). The third line shows the results with 6 QoS attributes: response time, availability, throughput, successability, reliability and compliance. In this case the query was submitted as (response time<=1000, availability>=95%, throughput>=20, sucessability>=95%, reliability >=80%, compliance>=85%) and consumer's preference vector is set as: (2, 1, 1, 2, 3, 3). The last row shows the execution time of each algorithm based on 9 QoS attributes: response time, availability, throughput, successability, reliability, compliance, best practice, latency and documentation. The query and preference vector in this case are (response time <=1000, availability>=95%, throughput>=20, sucessability>=95%, reliability>=80%, compliance>=80%, best practice>=50%, latency<=50ms), and (1,1,3,2,3,3,2,2,3) respectively. The average execution time was computed over 500 runs. Table 4.2 represents the performance of the algorithms on a dataset including 10 Web services. As indicated in Figure 4.2 the average execution time of DS_I is very close to that of DS algorithm.  We can also notice that the algorithms are relatively stable for different number of attributes. BF-Q is the slowest algorithm, but the difference between its processing time and that of the other algorithms is not that much.

Table 4.2- Execution time of algorithms on datasets including 10 web services and different number of attributes

| | **Average execution Time(ms)** | | | | |
|---|---|---|---|---|---|
| **Number of Attributes** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| 1 | 5 | 6 | 8 | 9 | 4 |
| 4 | 5 | 6 | 8 | 10 | 4 |
| 6 | 5 | 6 | 9 | 11 | 4 |
| 9 | 5 | 6 | 9 | 12 | 5 |

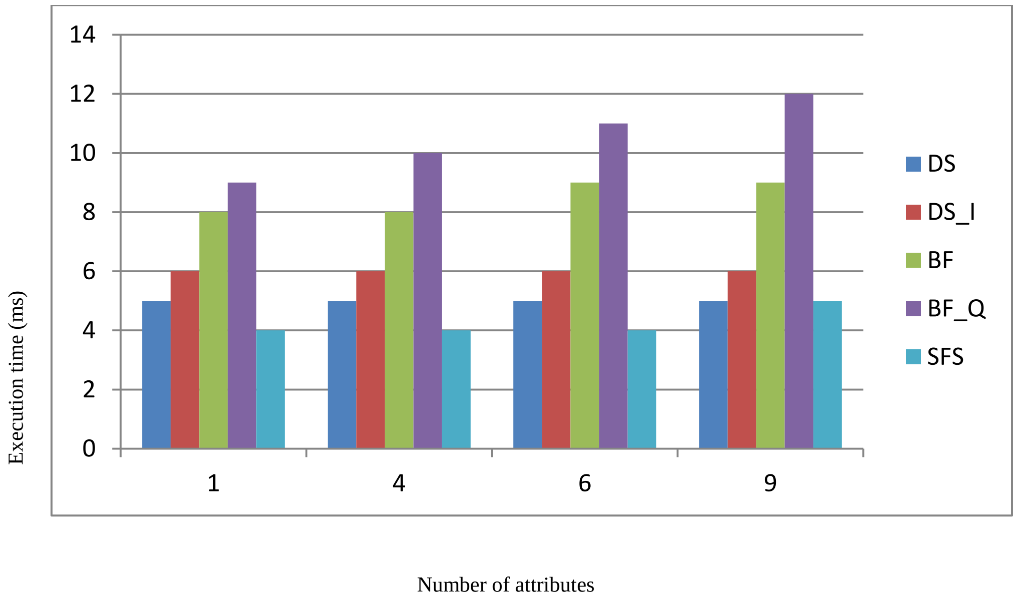

Figure 4.2- Comparison of execution time of 5 algorithms on datasets including 10 Web services and different number of attributes

In the next test we increase the size of the dataset. Table 4.3 compares the performance of algorithms on datasets including 50 Web services with different number of attributes. From

Figure 4.3 which was generated from Table 4.3, we can notice that SFS takes relatively less time than the other methods to generate the ranked list. We also can see that DS takes a very little time than DS_I to process the request, however the DS_I algorithms requires extra step to measure the distance between the services and the constraints specified in the query. According to the results, increasing the number of attributes does not affect the processing time of DS_I and BF_Q algorithms in large scale. Moreover, it is observed that the average execution time of BF_Q is close to that of BF. In this experiment SFS outperforms the other methods.

Table 4.3- Execution time of algorithms on datasets including 50 web services and different number of attributes

| | **Average execution Time (ms)** | | | | |
|---|---|---|---|---|---|
| **Number of Attributes** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| 1 | 21 | 22 | 27 | 29 | 15 |
| 4 | 22 | 23 | 27 | 30 | 15 |
| 6 | 22 | 23 | 28 | 31 | 15 |
| 9 | 22 | 23 | 28 | 31 | 16 |

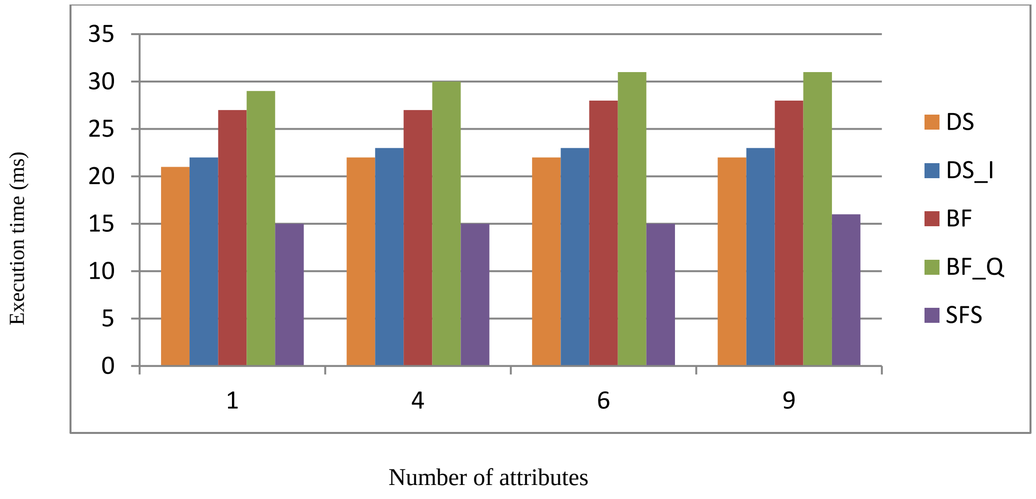

Figure 4.3- Comparison of execution time of 5 algorithms on datasets including 50 Web services and different number of attributes

We also conducted the same experiment with datasets containing 100 and 150 services with different number of attributes. However as all the results have the same pattern as the last two examples, we excluded them from this section. Please refer to Appendix A for further information.

To evaluate the result for larger sized datasets with more than 150 services, we first applied the algorithms on a dataset including 200 Web services with different number of attributes. The results are displayed in Table 4.4.

Table 4.4- Execution time of algorithms on datasets including 200 web services and different number of attributes

| | **Average execution Time(ms)** | | | | |
|---|---|---|---|---|---|
| **Number of Attributes** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| 1 | 78 | 79 | 82 | 85 | 75 |
| 4 | 78 | 79 | 82 | 85 | 82 |
| 6 | 79 | 79 | 84 | 88 | 98 |
| 9 | 79 | 80 | 84 | 90 | 114 |

We provide the comparison between performances of different approaches in Figure 4.4. The figure was generated from Table 4.4. It shows that SFS performs better that the other algorithms on the dataset when there is only one QoS attribute, however when the number of attributes is increased, DS and DS_I algorithms outperform SFS. We can also notice that even BF and BF_Q perform better than SFS when the number of attributes is greater than 4.

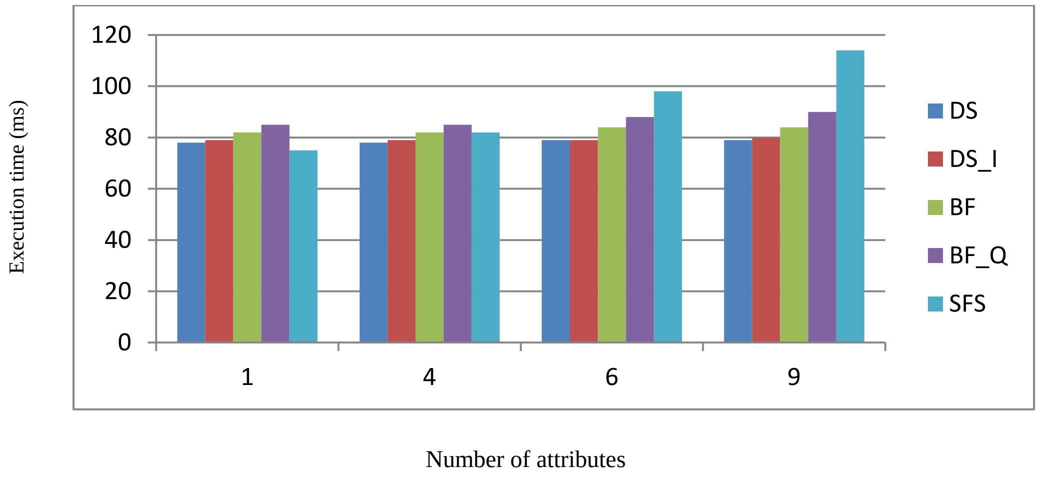

Figure 4.4- Comparison of execution time of 5 algorithms on datasets including 200 Web services and different number of attributes

Next, we increased the number of candidate Web services to 1000 and provided the average processing time of each algorithm in Table 4.5. The performance result of the algorithms is compared in Figure 4.5. Similar to the previous test, the results reveal that although the SFS runs faster when there is only one QoS attribute, its performance decreases dramatically when the number of attributes is increased. So it does not scale well when the number of attributes increases, whereas the performance of other 4 algorithms is very stable and is not affected by this change that much.

Table 4.5- Execution time of algorithms on datasets including 1000 web services and different number of attributes

| | **Execution Time** | | | | |
|---|---|---|---|---|---|
| **Number of Attributes** | **DS** | **DS-I** | **BF** | **BF_Q** | **SFS** |
| 1 | 392 | 392 | 415 | 435 | 286 |
| 4 | 395 | 395 | 418 | 437 | 545 |
| 6 | 398 | 398 | 418 | 440 | 798 |
| 9 | 403 | 410 | 420 | 431 | 1015 |

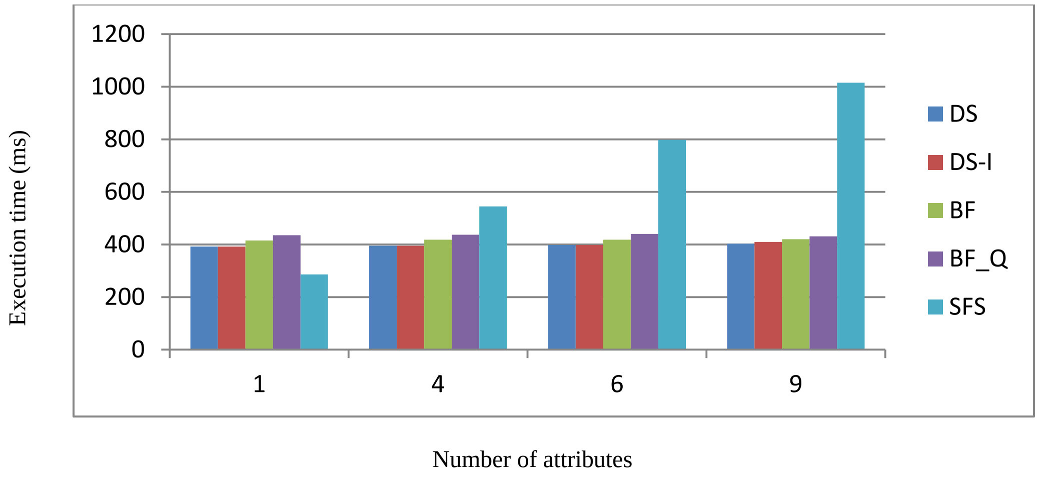

Figure 4.5- Comparison of execution time of 5 algorithms on datasets including 1000 Web services and different number of attributes

According to the observations, SFS has the lowest performance on the large sized datasets and large number of attributes. Moreover, it can be seen that the performance of DS and DS_I is relatively stable for all types of datasets. The growing number of the QoS attributes does not affect their processing time that much.

From all the tests done in experiment 2, we can conclude that SFS is a good option only for either very small sized datasets or a limited number of attributes. Furthermore DS_I algorithm is as effective as DS algorithm in all combination of datasets. We can also come to this conclusion that both DS and DS_I algorithms outweigh the other reviewed approaches for large sized datasets in terms of the efficiency.

**Experiment-3: Different Datasets with Different Types of Attributes**

The aim of the next set of experiments is to study how the different algorithms deal with attributes with different types. In this case we did the experiments by running the algorithms on different data sets with different number of Web services containing a combination of attributes with different types. Due to the space limit, we present only the result of the experiments with datasets including 50, 100 and 200 services. For further results, please refer to Appendix A.

The test was done in 2 steps: first we considered only one QoS attribute and then in the second step, we used datasets with five QoS attributes. In the first step, we considered "response-time" for numeric type, "throughput" for data interval type, "documentation" for Boolean type and "supported standards" for list type. In the next step, we conducted the experiments with 5 numeric QoS attributes: ("response time", "availability", "reliability", "successability", "latency"). To see how the algorithms behave when we add list, interval and Boolean types to the list of numeric QoS attributes, we generated list type values for attribute "supported standards"; interval data type values for attribute "throughput" and Boolean values for attribute "documentation". Then we added the generated data on top of 4 numeric attributes: ("response time", "availability", "reliability", "successability") in the second set of experiment to study the pattern of results for different data types. The results of both sets of tests are illustrated in Table 4.6.

Table 4.6- Execution time of algorithms on datasets with different type of attributes

| | Average execution Time (ms) | | | | | | |
|---|---|---|---|---|---|---|---|
| **Type of attribute** | **Number of services** | **Number of attributes** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| Numeric/Boolean | 50 | 1 | 21 | 22 | 27 | 29 | 15 |
| | | 5 | 22 | 23 | 27 | 30 | 15 |
| | 100 | 1 | 41 | 43 | 45 | 51 | 35 |
| | | 5 | 41 | 43 | 45 | 51 | 35 |
| | 200 | 1 | 78 | 79 | 82 | 85 | 75 |
| | | 5 | 78 | 79 | 84 | 87 | 84 |
| Data interval | 50 | 1 | 41 | 43 | 56 | 59 | 44 |
| | | 5 | 41 | 43 | 56 | 60 | 45 |
| | 100 | 1 | 60 | 63 | 68 | 71 | 64 |
| | | 5 | 60 | 63 | 69 | 73 | 105 |
| | 200 | 1 | 88 | 90 | 100 | 105 | 91 |
| | | 5 | 88 | 90 | 101 | 108 | 131 |
| list | 50 | 1 | 15 | 17 | 24 | 28 | 18 |
| | | 5 | 15 | 17 | 27 | 32 | 24 |
| | 100 | 1 | 31 | 34 | 40 | 45 | 35 |
| | | 5 | 33 | 36 | 43 | 49 | 38 |
| | 200 | 1 | 68 | 70 | 75 | 80 | 71 |
| | | 5 | 69 | 72 | 79 | 84 | 73 |

We displayed the results for a dataset including 50 Web services with 1 QoS attributes in Figure 4.6. We can observe how the interval data types affected the processing time of all algorithms. We can also see that although SFS is a good candidate for numeric and Boolean data types, its processing time for list and interval data type is higher than those of DS and DS-I algorithms. BF-Q and BF algorithms process the request for all data types with a slight difference. They are slower than the other algorithms due to the extra steps required for generating the individual ranked lists for each QoS attributes.

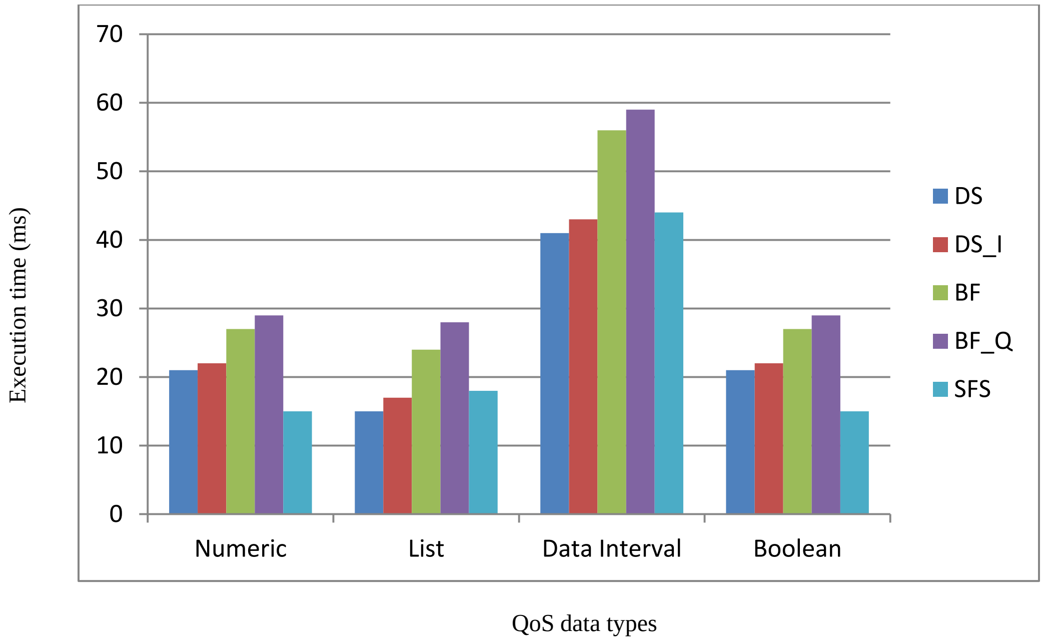

Figure 4.6- Comparison of execution time of 5 algorithms on dataset including 50 Web services and 1 QoS attribute with different data types

We excluded the graphical representation of the results for datasets including 50 Web services and 5 QoS attributes from this section, as they have the same pattern as shown in the previous figure. Please refer to Appendix A for further details.

In the next set of experiments we increased the number of Web services to 100 services. As the pattern of the results is the same as that of the previous experiment, we excluded the graph of observation for the dataset including 100 Web services and 1 QoS attribute. Please refer to Appendix A for further information.

Figure 4.7 shows the comparison of the performance of all algorithms for the combination of different type of attributes on a dataset including 100 Web services and 5 QoS attributes. Similar to the previous results, SFS still has better performance for numeric and Boolean data types. However its processing time for interval data types is even higher than those of BF and BF_Q algorithms. The efficiency of SFS is dramatically decreased for large sized datasets including one interval data type. According to the results, there is a small difference between the performance of BF and BF_Q.

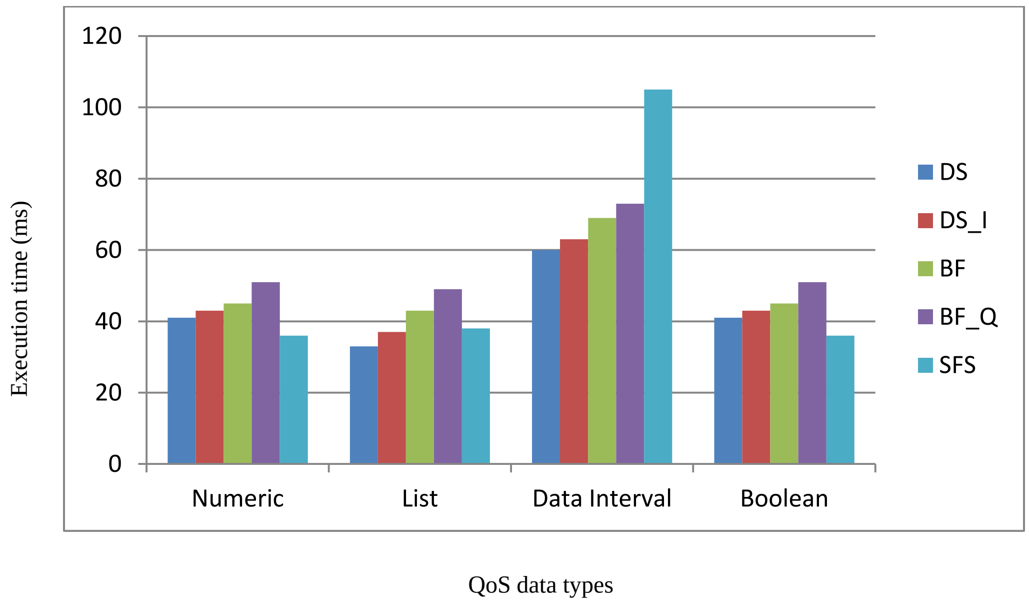

Figure 4.7- Comparison of execution time of 5 algorithms on dataset including 100 Web services and 5 QoS attributes with different data types

In the next set of test we increased the size of dataset to 200 services. As the results for 1 QoS attribute have the same trend as those shown in Figure 4.6, we excluded the graphical representation from this section. The detailed information is provided in Appendix A.

Figure 4.8 compares the average processing time of algorithm DS_I with the other algorithms on a dataset including 200 Web services and 5 QoS attributes. We notice that the performance of SFS has decreased for all data types and has the highest processing time for data interval type. With a small difference, DS and DS_I have the best performance in this experiment. As we discussed earlier, SFS is not a good candidate for large sized datasets. The performance of SFS decreases when we increase the number of QoS attributes. The results also indicate that efficiency of BF_Q algorithm is close to that of BF algorithm.

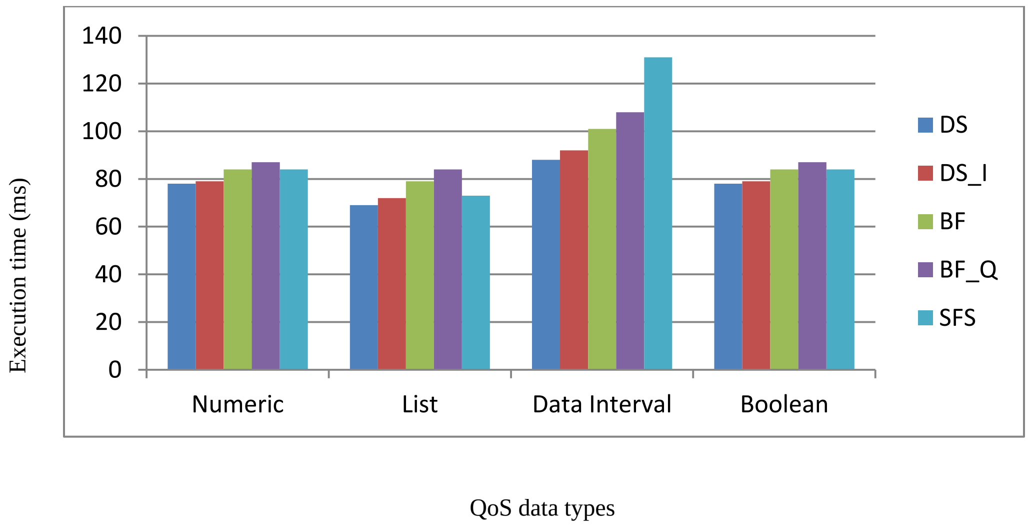

Figure 4.8- Comparison of execution time of 5 algorithms on dataset including 200 Web services and 5 QoS attributes with different data types

### 4.3.2. Quality of Results

In the following section we provide the evaluation analyses of the quality of ranking results by comparing the improved algorithms with the original algorithms and other approaches. Table 4.7 presents the URLs of 10 Web services with the equal functionality provided from

QWS. The Web services calculate specific national holidays in US. The quality attributes of the services are presented in Table 4.8.

Table 4.7- Dataset including 10 similar Web services

| | |
|---|---|
| $S_1$ | http://www.holidaywebservice.com/Holidays/GBEAW/Dates/GBEAWHolidayDates.asmx?WSDL |
| $S_2$ | http://www.holidaywebservice.com/Holidays/GBNIR/GBNIRHolidayService.asmx?wsdl |
| $S_3$ | http://www.27seconds.com/Holidays/HolidayService.asmx?WSDL |
| $S_4$ | http://www.holidaywebservice.com/Holidays/GBNIR/Dates/GBNIRHolidayDates.asmx?wsdl |
| $S_5$ | http://www.27seconds.com/Holidays/US/Dates/USHolidayDates.asmx?wsdl |
| $S_6$ | http://www.27seconds.com/Holidays/US/USHolidayService.asmx?wsdl |
| $S_7$ | http://www.holidaywebservice.com/Holidays/US/USHolidayService.asmx?WSDL |
| $S_8$ | http://www.holidaywebservice.com/Holidays/US/Dates/USHolidayDates.asmx?WSDL |
| $S_9$ | http://www.holidaywebservice.com/Holidays/HolidayService.asmx?WSDL |
| $S_{10}$ | http://www.holidaywebservice.com/Holidays/GBMBW/Dates/GBMBWHolidayDates.asmx?WSDL |

Table 4.8- 9 Quality attributes of Web services

| QoS | Response time(ms) | Availability % | Throughput | Successability % | Reliability % | Compliance % | Best Practice % | Latency (ms) | Documentation % |
|---|---|---|---|---|---|---|---|---|---|
| query | =<1000 | >=95 | >=14 | >=95 | >=80 | >=85 | >=80 | =<50 | >=50 |
| $S_1$ | 302.75 | 89 | 7.1 | 90 | 73 | 78 | 80 | 187.75 | 32 |
| $S_2$ | 482 | 98 | 16 | 95 | 73 | 100 | 84 | 1 | 2 |
| $S_3$ | 108.3 | 89 | 1.4 | 96 | 73 | 78 | 80 | 2.6 | 96 |
| $S_4$ | 126.17 | 98 | 12 | 100 | 67 | 78 | 82 | 22.77 | 89 |
| $S_5$ | 107 | 87 | 1.9 | 95 | 73 | 89 | 62 | 58.33 | 93 |
| $S_6$ | 107.57 | 80 | 1.7 | 81 | 67 | 78 | 82 | 18.21 | 61 |
| $S_7$ | 255 | 98 | 1.3 | 99 | 67 | 100 | 82 | 51 | 4 |
| $S_8$ | 136.71 | 76 | 2.8 | 76 | 60 | 89 | 69 | 11.57 | 8 |
| $S_9$ | 102.62 | 91 | 15.3 | 97 | 67 | 78 | 82 | 0.93 | 91 |
| $S_{10}$ | 93.37 | 96 | 13.5 | 99 | 67 | 89 | 58 | 41.66 | 93 |

We applied two algorithms DS and DS_I on the dataset for a query: (response time<=1000ms, availability>=95%, throughput>=14, successability>=95%, reliability>=80%, compliance>=85%, best-practice >=80%, latency<=50ms and documentation>=50%), and displayed the ranked lists results in Table 4.9. In this test we set the user's preferences on each QoS attribute as (1, 2, 3, 2, 3, 3, 2, 1, 3).

Table 4.9- Ranked results from DS and DS_I algorithms

| DS | | DS_I | |
|---|---|---|---|
| $s_4$ | http://www.holidaywebservice.com/Holidays/GBNIR/Dates/GBNIRHolidayDates.asmx? wsdl | $s_9$ | http://www.holidaywebservice.com/Holidays/HolidayService.asmx?WSDL |
| $s_9$ | http://www.holidaywebservice.com/Holidays/HolidayService.asmx?WSDL | $s_4$ | http://www.holidaywebservice.com/Holidays/GBNIR/Dates/GBNIRHolidayDates.asmx? wsdl |
| $s_3$ | http://www.27seconds.com/Holidays/HolidayService.asmx?WSDL | $s_3$ | http://www.27seconds.com/Holidays/HolidayService.asmx?WSDL |
| $s_{10}$ | http://www.holidaywebservice.com/Holidays/GBMBW/Dates/GBMBWHolidayDates.asmx?WSDL | $s_{10}$ | http://www.holidaywebservice.com/Holidays/GBMBW/Dates/GBMBWHolidayDates.asmx?WSDL |
| $s_7$ | http://www.holidaywebservice.com/Holidays/US/USHolidayService.asmx?WSDL | $s_2$ | http://www.holidaywebservice.com/Holidays/GBNIR/GBNIRHolidayService.asmx?wsdl |
| $s_5$ | http://www.27seconds.com/Holidays/US/Dates/USHolidayDates.asmx?wsdl | $s_7$ | http://www.holidaywebservice.com/Holidays/US/USHolidayService.asmx?WSDL |
| $s_2$ | http://www.holidaywebservice.com/Holidays/GBNIR/GBNIRHolidayService.asmx?wsdl | $s_5$ | http://www.27seconds.com/Holidays/US/Dates/USHolidayDates.asmx?wsdl |
| $s_6$ | http://www.27seconds.com/Holidays/US/USHolidayService.asmx?wsdl | $s_6$ | http://www.27seconds.com/Holidays/US/USHolidayService.asmx?wsdl |
| $s_1$ | http://www.holidaywebservice.com/Holidays/GBEAW/Dates/GBEAWHolidayDates.asmx?WSDL | $s_1$ | http://www.holidaywebservice.com/Holidays/GBEAW/Dates/GBEAWHolidayDates.asmx?WSDL |
| $s_8$ | http://www.holidaywebservice.com/Holidays/US/Dates/USHolidayDates.asmx?WSDL | $s_8$ | http://www.holidaywebservice.com/Holidays/US/Dates/USHolidayDates.asmx?WSDL |

According to the result returned from DS algorithm, $S_4$ obtains the higher ranking position in the ranked list than $S_9$, while in the ranked list obtained from DS_I, these two services appeared in an opposite order. The reason is that $S_4$ has the optimal values for two attributes: availability and successability, therefore it receives lower distance score than the other services, even though it does not satisfy the requirement specified for attribute "throughput". On the contrary, in results obtained from DS_I, $S_9$ receives a higher ranking order, as it was evaluated based on both its distance to optimal values and to the constraints specified by user. Service $S_7$ also appears in higher ranking position than $S_2$ in the list returned by DS algorithm, whereas it fails to satisfy the requirements specified for attribute "latency" and "throughput". However service $S_2$ satisfies most of the user's requirements and is strictly better than service $S_7$ in terms of some other qualities. So we could see that our improved algorithm actually takes into account the actual user requirement so that a different ranking order could be generated for a different user request. The user's demand is an essential feature for the selection task, however missing in current solutions.

We also applied BF and BF_Q algorithms on the same dataset and requirements. The ranked results are displayed in Table 4.10. In the result list retrieved from BF, service $S_1$ has higher ranking position than service $S_6$ , whereas they appeared in an opposite order in the result list retrieved from BF_Q algorithm. Service $S_1$ meets none of the requirements specified in the query, except the demand on attribute "response time" and "best practice", nevertheless it appears in higher position than service $S_6$ which meets the requirements specified for attributes "response time", "documentation", "latency" and "best practice" and is similar to $S_1$ over other attributes.

From this case study we notice that how ignoring the user's requirements could impact the ranking orders of the candidates in the result list. For larger sized datasets, many appropriate services might be placed in the lower ranking positions or even excluded from the list, and therefore they miss the chances to be selected by users.

Table 4.10- Ranked results from B and BF_Q algorithms

| **BF** | | **BF_Q** | |
|---|---|---|---|
| $s_4$ | http://www.holidaywebservice.com/Holidays/GBNIR/Dates/GBNIRHolidayDates.asmx? wsdl | $s_4$ | http://www.holidaywebservice.com/Holidays/GBNIR/Dates/GBNIRHolidayDates.asmx? wsdl |
| $s_9$ | http://www.holidaywebservice.com/Holidays/HolidayService.asmx?WSDL | $s_9$ | http://www.holidaywebservice.com/Holidays/HolidayService.asmx?WSDL |
| $s_2$ | http://www.holidaywebservice.com/Holidays/GBNIR/GBNIRHolidayService.asmx?wsdl | $s_2$ | http://www.holidaywebservice.com/Holidays/GBNIR/GBNIRHolidayService.asmx?wsdl |
| $s_{10}$ | http://www.holidaywebservice.com/Holidays/GBMBW/Dates/GBMBWHolidayDates.asmx?WSDL | $s_{10}$ | http://www.holidaywebservice.com/Holidays/GBMBW/Dates/GBMBWHolidayDates.asmx?WSDL |
| $s_3$ | http://www.27seconds.com/Holidays/HolidayService.asmx?WSDL | $s_3$ | http://www.27seconds.com/Holidays/HolidayService.asmx?WSDL |
| $s_5$ | http://www.27seconds.com/Holidays/US/Dates/USHolidayDates.asmx?wsdl | $s_7$ | http://www.holidaywebservice.com/Holidays/US/USHolidayService.asmx?WSDL |
| $s_7$ | http://www.holidaywebservice.com/Holidays/US/USHolidayService.asmx?WSDL | $s_5$ | http://www.27seconds.com/Holidays/US/Dates/USHolidayDates.asmx?wsdl |
| $s_1$ | http://www.holidaywebservice.com/Holidays/GBEAW/Dates/GBEAWHolidayDates.asmx?WSDL | $s_6$ | http://www.27seconds.com/Holidays/US/USHolidayService.asmx?wsdl |
| $s_6$ | http://www.27seconds.com/Holidays/US/USHolidayService.asmx?wsdl | $s_1$ | http://www.holidaywebservice.com/Holidays/GBEAW/Dates/GBEAWHolidayDates.asmx?WSDL |
| $s_8$ | http://www.holidaywebservice.com/Holidays/US/Dates/USHolidayDates.asmx?WSDL | $s_8$ | http://www.holidaywebservice.com/Holidays/US/Dates/USHolidayDates.asmx?WSDL |

In order to evaluate the difference between the improved algorithm and the original model, we used Kendall tau distance metric [41], [42]. Let S= $\{s_1, s_2, \ldots, s_n\}$ as an input to two different ranking algorithms. Suppose $L_1$ and $L_2$ are two ranked lists returned by each algorithm. Kendall tau measures the rank correlation between the lists by counting the number of pair-wise disagreements between the items of the lists. The Kendall tau correlation coefficient ($\tau$), and the Kendall tau distance (*K*) is measured as follows:

$$\tau(L_1, L_2) = \frac{(\text{number of concordant pairs - number of discordant pairs})}{\frac{1}{2}n(n-1)} \qquad (4.1)$$

$$K\,(L_1, L_2) = \frac{\text{number of discordant pairs}}{\frac{1}{2}n(n-1)} \qquad (4.2)$$

Kendall tau distance lies in the interval [0, 1]. The Kendall tau distance is 0 if the items of each list are in the same order and is 1 if all the items of the lists are in the reverse orders. Kendall tau correlation coefficient lies in the interval [-1, 1]. The coefficient is 1 if the ranking lists are the same and is -1 if one ranking list is the reverse of the other one. According to [43], the value of calculated correlation between [0, 0.2] shows a very weak correlation or similarity, values between [0.21, 0.4] represent a weak correlation, values between [0.41, 0.60] indicates a moderate correlation, [0.61, 0.80] shows a strong correlation and values in data range [0.81, 1] shows a very strong correlation. We also studied the trend of changes in the quality of the results based on datasets with different sizes. To accomplish this, we considered Distance-based algorithm which is a fast and reliable method as the baseline for our comparison in terms of the quality of the results. Our goal is to show how the results change when the query requirements are employed in the model. To compare the results, we applied the algorithms to different datasets from QWS. We used an online Kendall tau calculator provided by Wessa [44] to estimate the correlation coefficient values. In order to compare DS and DS_I algorithms, we performed three sets of tests. In the first experiment, we supposed that both query constraints and optimal values play equal roles in the ranking process. So we set $W_1$=0.5 and $W_2$=0.5 in equation (3.14). In the next experiment we set the bigger weight for the scores received by considering query constraints in the algorithm and set the weights as $W_1$=0.7 and $W_2$=0.3. In the last experiment we considered a larger weight for the optimal value as $W_1$=0.3 and $W_2$=0.7. Table 4.11 shows all the computed correlation coefficient values for each algorithm on different sized datasets in each experiment.

Table 4.11-Comparison of Kendall tau correlation coefficient between DS_I and original DS algorithm

| | **Kendall tau values** | | |
|---|---|---|---|
| | **Test 1** | **Test 2** | **Test 3** |
| Number of Web Services | **DS_I** ($W_1$=0.5, $W_2$=0.5) | **DS_I** ($W_1$=0.7,$W_2$=0.3) | **DS_I** ($W_1$=0.3, $W_2$=0.7) |
| 10 | 0.86 | 0.81 | 0.99 |
| 50 | 0.89 | 0.84 | 0.92 |
| 100 | 0.88 | 0.84 | 0.94 |
| 150 | 0.89 | 0.86 | 0.94 |
| 200 | 0.88 | 0.85 | 0.94 |
| 250 | 0.88 | 0.86 | 0.95 |
| 300 | 0.89 | 0.84 | 0.94 |
| 500 | 0.87 | 0.84 | 0.95 |
| 1000 | 0.88 | 0.85 | 0.94 |
| 1500 | 0.87 | 0.84 | 0.95 |
| 2000 | 0.89 | 0.84 | 0.95 |

As a result we can observe that the Kendall tau correlation values decreased when a bigger weight was assigned to query constraints. In this case, the results show the higher dissimilarity between the original algorithm and the improved approach. It is possible that there might be some services in the dataset that meet all of the requirements specified by the user, but they get lower overall ranking score in the original method and consequently appear in the lower positions in the final ranked list.

We also computed the Kendal tau correlation coefficient values between other algorithms and DS_I algorithm. The results are presented in Table 4.12. According to the results, we can notice that the ranking results retrieved from SFS approach is very similar to the results returned by DS_I algorithm, however as we had already noticed, DS_I is more efficient and stable on large sized datasets. We also noticed that DS_I algorithm is relatively as efficient as SFS on small sized datasets. As a result DS_I can be considered an efficient method to be integrated in different real-time Web service discovery and selection systems.

BF and BF_Q approaches have the smallest similarity degree with DS_I, as they rely only on the positional value of a service in each individual ranked list. They do not consider a balanced similarity degree between the quality of a service and the requirement specified in a query.

Table 4.12- Comparison of Kendall tau correlation coefficient between BF, BF_Q, SFS and DS_I algorithm

| Number of Services | **BF** | **BF_Q** | **SFS** |
|---|---|---|---|
| 10 | 0.75 | 0.75 | 0.90 |
| 50 | 0.73 | 0.77 | 0.81 |
| 100 | 0.68 | 0.68 | 0.79 |
| 150 | 0.65 | 0.68 | 0.79 |
| 200 | 0.66 | 0.68 | 0.79 |
| 250 | 0.65 | 0.69 | 0.79 |
| 300 | 0.67 | 0.69 | 0.79 |
| 500 | 0.67 | 0.69 | 0.78 |
| 1000 | 0.57 | 0.56 | 0.79 |
| 1500 | 0.57 | 0.56 | 0.78 |
| 2000 | 0.57 | 0.56 | 0.78 |

In this section we compared the results of the improved algorithms with the original models and SFS to show how the results vary when we consider user's requirement in the algorithms. However it is hard to measure which one has better quality because to the best of our knowledge, there is still no standard way to compare the quality of different ranking lists in a Web service discovery system. Besides it is a subjective opinion depending on different people and different scenarios and different people consider different scales for evaluating the results.

## 4.4. Chapter Summary

In this chapter we explained our implementation of the proposed improved algorithms and the baseline algorithms, as well as the experiment design and the datasets we used. Then we presented the results of our experiment of conducting various tests and provided some analyses on the final ranked lists. To measure the efficiency of the improved algorithm, we computed the average execution time of each algorithm by using various datasets with different number of QoS attributes from different data types. Then we compared the improved algorithms with the original methods in terms of the average processing time. Finally we compared the quality of the results of the improved algorithm with the original one to study the impact of user's demand on the results.

# CHAPTER 5
# CONCLUSION AND FUTURE WORK

## 5.1. Conclusion

In this thesis, we reviewed some of the ranking models for QoS-based Web service selection system such as Borda Fuse, Skyline operation and Distance-based algorithms. We addressed the issues related to each algorithm and then proposed a solution by presenting a model inspired by Distance-based method. The advantage of the improved method is to address the deficiency of other approaches caused by ignoring the role of user's requirements in their ranking process. Different from the majority of the current works, in this method both the optimal value of each QoS attribute, and also the constraints specified in a user's query are taken into account. We implemented this model by using Euclidean metric to measure the similarity between an offered service and the service with optimal values. Besides we computed the similarity degree between the service and the constraints required by a user. The final ranking score is the aggregation of both similarity scores.

Considering user's demand, we extended Borda Fuse method to deal with different user's requirements. We considered this method to study how different algorithms with simple structures deal with user’s requirements. To improve the Borda Fuse algorithm, we added the query vector to list of candidate services in order to evaluate whether they could satisfy the user queries, and then computed the final ranking score based on the summation of the positional values of each service in each ranking list based on each QoS attribute. In order to validate the framework and compare the optimized algorithms with other models, we conducted extensive experiments on various datasets with different specifications. By increasing the size of datasets and the number of QoS attributes, we compared all algorithms in terms of the processing time. According to the observations, SFS is the fastest algorithm when we have only small sized dataset and one QoS attribute. By increasing the size of the dataset and the number of attributes, SFS runs slower. On the contrary DS and DS_I are the fastest algorithms on the large sized datasets. BF and BF_Q run faster than SFS on large sized datasets with larger number of QoS attributes. We also performed different experiments with different type of attributes such as numeric, Boolean, list and data interval. We noticed that SFS has a poor performance on attributes with data interval type. DS and DS_I with a slight different in execution time have the best performance for all data types when the size of the dataset is large.

We also compared the quality of results of DS_I method with that of DS, and noticed that when we set a higher weight to query constraints, the dissimilarity between the results is larger. If we consider equal weight for consumer's requirements and optimized services, then we receive more similar results. As a result it can be left in consumer's hand to decide if the ranking system should focus on query constraints or optimized services.

## 5.2. Main Contributions

There are three main contributions of this thesis:

- We provided an enhancement ranking algorithm based on a Vector-based model which is capable of dealing with user's requirements and measuring the ranking relation between services efficiently.
- We improved a rank aggregation based algorithm (Borda Fuse) to cover the user's requirement and provide more accurate results.

- We compared the enhanced algorithms with one of the well-accepted ranking algorithms (SFS) to show how they are more efficient on large sized datasets with large number of attributes and different data types.

## 5.3. Future Work

As future work, we can consider the following directions:

(1) Improve the framework to support top-*k* query processing effectively. Users might be interested in being presented with only a limited number of best options based on their requirements. In the other word, we could focus on extending the model to be able to return the *k* best results based on the user's requirements. In this case the processing time could be much lower, and users would be able to select their desired services easily.

(2) To devise a user interface model for the service selection system and integrate the ranking algorithm with the model to provide an easier method for user to select the desired services based on his requirements.

(3) Improve the algorithm to support imprecise QoS dimensions by using fuzzy sets. In our work we included different types of QoS attributes, but to be more flexible and compatible with all kind of QoS parameters, we also need to include fuzzy sets in the framework.

(4) To improve the algorithm to include a systemized method for assigning weights for the scores based on both optimal values and user's requirements. In this case the system would by more user friendly as users are not obliged to assign the weights.

# APPENDIX A- RESULTS FROM EXPERIMENTS

Table A.1-Execution time of algorithms on datasets including 150 web services and different number of attributes

| | Average execution Time (ms) | | | | |
|---|---|---|---|---|---|
| Number of Attributes | DS | DS_I | BF | BF_Q | SFS |
| 1 | 59 | 22 | 27 | 29 | 50 |
| 4 | 59 | 23 | 27 | 29 | 51 |
| 6 | 60 | 23 | 28 | 31 | 59 |
| 9 | 60 | 61 | 67 | 74 | 69 |

Table A.2- Execution time of algorithms on datasets including 500 web services and different number of attributes

| | Average execution Time (ms) | | | | |
|---|---|---|---|---|---|
| Number of Attributes | DS | DS_I | BF | BF_Q | SFS |
| 1 | 198 | 200 | 210 | 220 | 167 |
| 4 | 198 | 201 | 211 | 221 | 232 |
| 6 | 199 | 201 | 211 | 222 | 280 |
| 9 | 199 | 201 | 214 | 224 | 301 |

Table A.3- Execution time of algorithms on datasets including 1500 web services and different number of attributes

| | **Average execution Time(ms)** | | | | |
|---|---|---|---|---|---|
| **Number of Attributes** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| 1 | 638 | 638 | 701 | 715 | 446 |
| 4 | 640 | 641 | 703 | 718 | 770 |
| 6 | 640 | 641 | 703 | 718 | 883 |
| 9 | 643 | 646 | 709 | 723 | 1015 |

Table A.4- Execution time of algorithms on datasets including 2000 web services and different number of attributes

| | **Average execution Time (ms)** | | | | |
|---|---|---|---|---|---|
| **Number of Attributes** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| 1 | 870 | 871 | 995 | 1018 | 680 |
| 4 | 870 | 871 | 997 | 1020 | 885 |
| 6 | 871 | 873 | 998 | 1020 | 910 |
| 9 | 873 | 887 | 1005 | 1029 | 1215 |

Table A.5- Execution time of algorithms on different sized datasets with different type of attributes

| | Average execution Time (ms) | | | | | | |
|---|---|---|---|---|---|---|---|
| **Type of attribute** | **Number of services** | **Number of attributes** | **DS** | **DS_I** | **BF** | **BF_Q** | **SFS** |
| Numeric/Boolean | 250 | 1 | 94 | 95 | 106 | 110 | 83 |
| | | 5 | 94 | 95 | 107 | 111 | 113 |
| | 500 | 1 | 198 | 200 | 210 | 220 | 167 |
| | | 5 | 198 | 201 | 210 | 220 | 227 |
| | 1000 | 1 | 392 | 392 | 414 | 435 | 286 |
| | | 5 | 392 | 392 | 415 | 436 | 645 |
| Data interval | 250 | 1 | 173 | 178 | 184 | 191 | 184 |
| | | 5 | 173 | 178 | 185 | 191 | 199 |
| | 500 | 1 | 192 | 200 | 208 | 215 | 205 |
| | | 5 | 192 | 201 | 209 | 215 | 225 |
| | 1000 | 1 | 210 | 217 | 225 | 235 | 221 |
| | | 5 | 210 | 219 | 226 | 235 | 241 |
| list | 250 | 1 | 92 | 94 | 100 | 103 | 95 |
| | | 5 | 92 | 94 | 103 | 107 | 98 |
| | 500 | 1 | 187 | 188 | 197 | 207 | 191 |
| | | 5 | 187 | 188 | 199 | 209 | 196 |
| | 1000 | 1 | 390 | 392 | 401 | 410 | 395 |
| | | 5 | 390 | 392 | 404 | 415 | 453 |

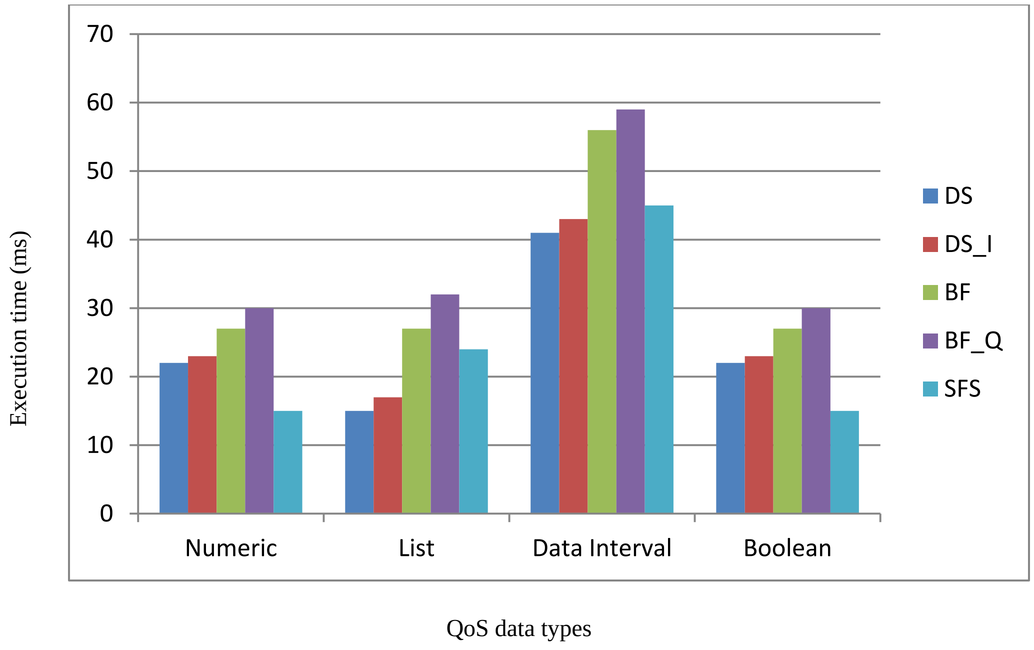

Figure A.1-Comparison of execution time of 5 algorithms on dataset including 50 Web services and 5 QoS attributes with different data types

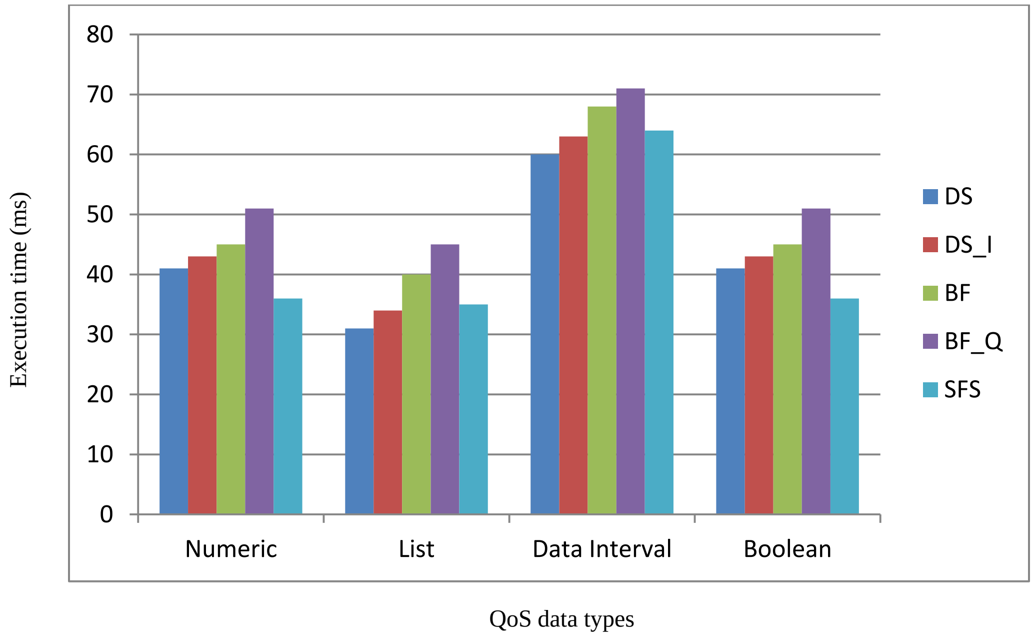

Figure A.2- Comparison of execution time of 5 algorithms on dataset including 100 Web services and 1 QoS attribute with different data types

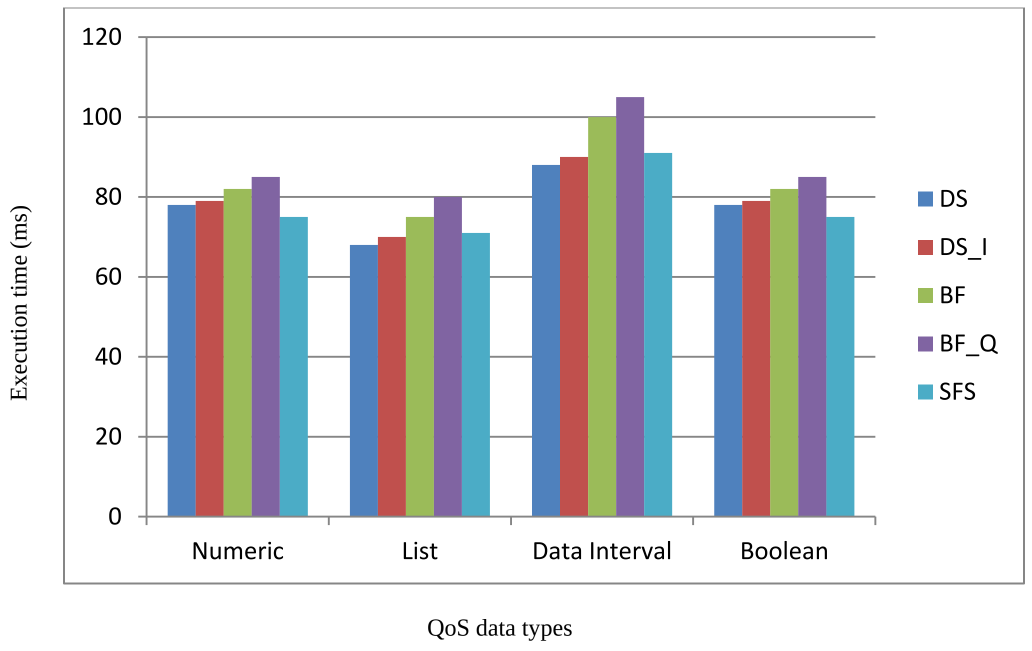

Figure A.3-Comparison of execution time of 5 algorithms on dataset including 200 Web services and 1 QoS attribute with different data types

# REFERENCES

[1] T. Earl, "SOA: Principles of Service Design", 2nd Edition. Upper Saddle River, N.J., Prentice-Hall, ISBN: 9780132344821, 2008.

[2] UDDI, "UDDI Technical White Paper", Sept. 2000 [cited July.9, 2011], available from Word Wide Web: <http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf>.

[3] W. Rong, K. Liu, and L.Liang, "Towards Personalized Ranking in Web Service Selection", in *Proceedings of the IEEE International Conference on e-Business Engineering*, Xi'an, China, pp.165-172, 2008.

[4] D. Bianchini, V.De Antonellis, and M. Melchiori, "QoS in ontology-based service classification and discovery", in *Proceedings of the 15th International Workshop on Database and expert Systems Applications*, Zaragoza, pp. 145-150, 2004.

[5] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, and S. McIlraithm, "OWL-S: Semantic Markup for Web Services" [online], Nov. 2004 [cited Aug 6, 2011], available from World Wide Web: < http://www.w3.org/Submission/OWL-S/>.

[6] K. Kritikos, and D. Plexousakis, "Semantic QoS Metric Matching", in *Proceedings of the ECOWS 2006: 4th European Conference on Web Services*, Zurich , pp. 265-274, 2006.

[7] E. Giallonardo, E. Zimeo, "More Semantics in QoS Matching", *in Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, Newport Beach, CA, pp. 163-171, 2007.

[8] I. Toma1, D. Roman, D. Fensel, B. Sapkota, and J.M. Gomez, "A Multi-criteria Service Ranking Approach Based on Non-Functional Properties Rules Evaluation", in *Proceedings of the Fifth International Conference on Service- Oriented Computing*, Vienna, pp. 435-441, 2007.

[9] O. Mola, P. Emamian, and M. Razzazi, "A Vector Based Algorithm for Semantic Web", in *Proceedings of the 3rd International Conference on Information and Communication Technologies:From Theory to Applications*, Damascus, pp. 1-5,2008.

[10] C. Zhou, L.T. Chia, and B.S. Lee, "Web Services Discovery with DAML-Qos Ontology", *International Journal of Web Services Research*, vol. 2, no. 2, pp. 44-67, 2005.

[11] P.V. Hentenryck, and V. Saraswat, "Strategic Directions in Constraint Programming", *Journal of ACM Computing Surveys* , vol. 28, no. 4, pp. 701-726, 1996.

[12] A. Ruiz-Cortés, O. Martín-Díaz, A.D. Toro, and M. Toro, "Improving the Automatic Procurement of Web Services Using Constraint Programming", *International Journal on Cooperative Information Systems*, vol. 14, no. 4, pp. 439-468, 2005.

[13] L. Sha, "A QoS based Web Service Selection Model", in *Proceedings of the International Forum on Information Technology and Applications*, Chengdu, pp. 353-356, 2009.

[14] J. Yan, and J. Piao, "Towards QoS-based Web Service Discovery", in *Proceedings of the International Conference on Service Oriented Computing*, Sydney, pp. 200-210, 2009.

[15] Y. Liu, and H. He, "Grid Service Selection Using QoS Model", in *Proceedings of the Third International conference on Semantics, Knowledge and Grid*, Xi'an, Shaanxi, pp. 576-577, 2007.

[16] S. Degwekar, S.Y.W. Su, and H. Lam, "Constraint Specification and Processing in Web Services Publication and Discovery", in *Proceedings of the IEEE International Conference on Web Services* , San Diego, CA , pp.210-217, 2004.

[17] P. Li, M. Comerio, A. Maurino, and F. De Paoli, "Advanced Non-functional Property Evaluation ofWeb Services", in *Proceedings of the Seventh IEEE European Conference on Web Services*, Eindhoven, pp. 27-36, 2009.

[18] K. Kritikos, and D. Plexousakis, "Semantic QoS-based Web Service Discovery Algorithms", in *Proceedings of the Fifth European Conference on Web Services*, Halle, pp. 181-190, 2007.

[19] P. Li, M. Comerio, A. Maurino, and F.D. Paoli, "Advanced Non-functional Property Evaluation of Web Services", in *Proceedings of the 7th IEEE European Conference on Web Services* , Eindhoven, pp. 27-36, 2009.

[20] K. Kritikos, and D. Plexousakis, "Mixed-Integer Programming for QoS-Based Web Service Matchmaking", *IEEE Transaction on Services Computing*, vol. 2, no. 2, pp.122-139, 2009.

[21] C. Herssens, I.J. Jureta, and S. Faulkner, "Dealing with Quality Tradeoffs during Service Selection", in *Proceedings of the 5th International Conference on Autonomic Computing*, Chicago, IL, pp. 77-86, 2008.

[22] V.X. Tran, H. Tsuji, and R. Masuda, "A New QoS Ontology and its QoS-based Ranking Algorithm for Web Services", *Journal of Simulation Modeling Practice and Theory*, vol. 17, no. 8, pp. 1378-1398, 2009.

[23] J. A. Aslam, and M. Montague, "Models for Metasearch", in *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, pp. 276-284, 2001.

[24] B. T. Bartell, G. W. Cottrell , and R.K. Belew, "Automatic combination of multiple ranked retrieval system", in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval,* New York, NY , pp. 173-181, 1994.

[25] R. Fagin, R. Kumar, and D. Sivakumar, "Efficient Similarity Search and Classification via Rank Aggregation", in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data,* New York, NY , pp. 950-961, 2003.

[26] Y. Yao, X. Chen, and S. Zhu, "Rank Aggregation Algorithms Based on Voting Model for Metasearch", in *Proceedings of the 2006 International Conference on Wireless Communications, Networking and Mobile Computing*, Wuhan , pp. 1-4, 2006.

[27] P. Diaconis, and R.Graham, "Spearman’s Footrule as a Measure of Disarray", *Journal of the Royal Statistical Society*, pp. 262-268, 1977.

[28] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank Aggregation Methods for the Web", in *Proceedings of the 10th World Wide Web Conference,* New York, NY, pp. 613-622, 2001.

[29] M. Montague, and J. A. Aslam, "Condorcet Fusion for Improved Retrieval", in *Proceedings of the 11th International Conference on Information and knowledge management,* New York, NY, pp. 538-548, 2002.

[30] D. Kossmann , S. Borzsony, and K. Stocker, "The Skyline Operator", in *Proceedings of the 17th International Conference on Data Engineering*, Heidelberg, pp. 421-430 , 2001.

[31] D. Papadias, Y. Tau, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems", *ACM Transactions on Database Systems (TODS)* , vol. 30, no. 1, pp. 41-82, 2005.

[32] J. Chomicki, B. Godfrey, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting", in *Proceedings of the 19th International Conference on Data Engineering*, Bangalore, pp. 71-719, 2003.

[33] J.Chomicki, B. Godfrey, P. Godfrey, J. Gryz, and D. Liang, "Skyline with Presorting", Department of Computer Science ,Toronot, Ont., Report No. CS-2002-04, Oct. 2002.

[34] H. Han, H. Jung, S. Kim, and H.Y. Yeom, "A Skyline Approach to the Matchmaking Web Service", in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid,* Shanghai, China, pp. 436-443, 2009.

[35] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and Clustering Web Services Using Multicriteria Dominance Relationships", *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 163-177, 2010.

[36] A. Sengupta, and T.K. Pal, "On Comparing Interval Numbers", *European Journal of Operational Research*, vol.127, no. 1, pp. 28-43, 2000.

[37] J. Hu, C. Guo, H. Wang, and P. Zou, "Quality Driven Web Services Selection", in *Proceedings of the IEEE International Conference on e-Business Engineering* , 681-685, 2005.

[38] Y.M. Wang, and T.M.S. Elhag, "On the Normalization of Interval and Fuzzy Weights", *Journal of Fuzzy Sets and Systems*, vol. 157, no. 18, pp. 2456-2471, 2006.

[39] E. Al-Masri, and Q.H. Mahmoud, "Discovering the Best Web Service", in *Proceedings of the 16th International Conference on World Wide Web*, Banff, AB , pp. 1257 – 1258, 2007.

[40] E. Al-Masri, Q.H. Mahmoud, "QoS-based Discovery and Ranking of Web Services", in *Proceedings of the 16th International Conference on Computer Communications and Networks*, Honolulu, HI, pp. 529 – 534, 2007.

[41] M.G. Kendall, "A new measure of rank correlation", *Biometrika journal*, vol.30, no.1-2, pp. 81-93, 1938.

[42] R.B. Nelsen, "Kendall tau metric", in *Hazewinkel Michiel Encyclopaedia of Mathematics*, Springer, ISBN 978-1556080104, 2001.

[43] J. Cohen, "*Statistical power analysis for the behavioral sciences*", 2nd Edition, Taylor & Francis, Inc, ISBN: 0805802835, 1988.

[44] P. Wessa, "Multivariate Correlation Matrix in Free Statistics Software", Office for Research Development and Education, available at: ttp://www.wessa.net/Patrick.Wessa/rwasp_pairs.wasp, last retrieved in July 2011.