PERSONALIZED DECISION MAKING FOR QOS-BASED WEB SERVICE SELECTION

By

Muhammad Suleman Saleem

M.Sc. in Computer Science, University of Central Punjab, Pakistan, 2000

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, 2014

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final versions, as accepted by my examiners.

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

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

I understand that my thesis may be made electronically available to the public.

MUHAMMAD SULEMAN SALEEM

PERSONALIZED DECISION MAKING FOR QOS-BASED WEB SERVICE SELECTION

Muhammad Suleman Saleem

Master of Science, Computer Science, 2014

Ryerson University

ABSTRACT

In order to choose from a list of functionally similar services, users often need to make their decisions based on multiple QoS criteria they require on the target service. In this process, different users may follow different decision making strategies, some are compensatory and some are non-compensatory. Most of the current QoS-based service selection systems do not consider these decision strategies in the ranking process, which we believe are crucial for generating accurate ranking results for individual users. In this thesis, we propose a decision strategy based service ranking model. Furthermore, considering that different users follow different strategies in different contexts at different times, we apply a learning to rank algorithm to learn a personalized ranking model for individual users based on how they select services in the past. Our experiment result shows the effectiveness of the proposed approach.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Dr. Cherie Ding for her valuable support and guidance in helping me to go through all the difficulties in my work. Her precious suggestions and guidance have greatly enhanced my knowledge and skills in research and have significantly contributed to the completion of this thesis.

In addition, I would like to thank Dr. Alex Ferworn, Dr. Alireza Sadeghian and Dr. Ali Miri who have reviewed my thesis and have given me valuable comments which enabled me to improve my thesis.

Also, I would like to acknowledge the support of the Computer Science Department of Ryerson University and my fellow students.

Finally, I would like to express my deep appreciations to my family, relatives, and friends who have motivated and supported me during these years of study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

AHP: Analytic Hierarchy Process

CP: Constraint Programming

EBA: Elimination by Aspects

EQW: Equal Weight

JND: Just Noticeable Difference

LEX: Lexicographic

MAP: Mean Average Precision

MCD: Majority of Confirming Dimensions

MCDM: Multi-Criteria Decision Making

MIP: Mixed Integer Programming

MRR: Mean Reciprocal Rank

NDCG: Normalized Discounted Cumulated Gain

QoS: Quality of Service

RR: Reciprocal Rank

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

UDDI: Universal Description, Discovery and Integration

WADD: Weighted Additive

WMCD: Weighted Majority of Confirming Dimensions

WSDL: Web Service Description Language

XML: Extensible Markup Language

CHAPTER 1

INTRODUCTION

1.1 Background and the Problem Statement

1.1.1 Background

Since the explosion of World Wide Web in the 1990s, the Internet has become an increasingly significant information source for users. Users can use Internet for finding any type of information over the web and can browse and exchange information. In the past, we were using web of data for finding the information over the web but with the passage of time we are moving from web of data to web of services because web services are enhancing the capabilities of web of pages. Now the web is giving more and more access to web of services. They are the software entities that are reachable over the Internet and are designed to accomplish the specific tasks. They can be used in multiple ways and can be flexibly incorporated both in traditional software systems and in web pages.

Web services are self-contained, loosely coupled, discoverable, autonomous and dynamic entities that are available in the network. They are used to support the development of fast growing, reusable, low cost and interoperable software and applications[33][34]. Web services construction is established according to common standards that ensures the successful interaction between service providers, service requestors and service brokers.

They provide a machine to machine communication over a network by using a series of standardized technologies, including WSDL (Web Services Description Language), SOAP (Simple Object Access Protocol) and UDDI (Universal Description, Discovery and Integration)

[35], where UDDI is designed to be interrogated by SOAP messages, as well as to provide an access to WSDL documents which describe the protocol bindings and description of web services [36]. Here SOAP is a XML based communication protocol which acts as a vehicle for carrying the information between computers or applications.

Web services involve the process of publishing, locating and accessing across the web. The main objects involved in this process are Service Providers, Service Directory and Service Consumers. These objects do not work independently, as they are interrelated with each other. Here service provider is the provider of the web service. It implements the service and publishes it to the service directory. The service consumer is the one who requests the required web service through a XML request. Service directory is the centralized directory of web services. It provides a central place where service providers can publish their services and service consumers can find their services. Figure 1.1 shows the basic web services architecture.

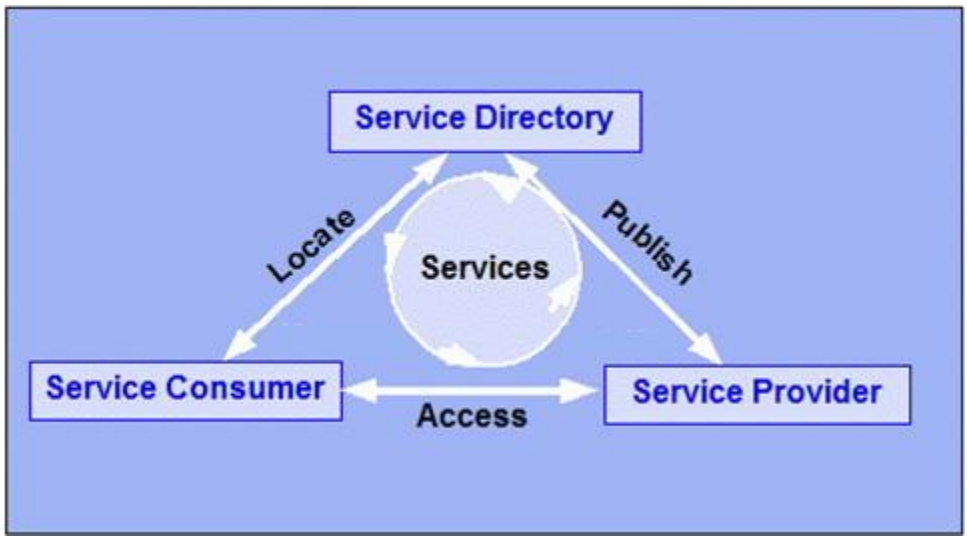

Figure 1.1 – Basic web services architecture [37].

Because of the large number of web services, it is very difficult to discover and select the required services. Therefore discovery and selection of web services is an important challenge for the web service community. In the first phase which is the discovery phase, the user searches

for a particular web service from the service directory or service registries and this search is based on the user's functional requirement. Functional requirements of web services determine the overall behavior of web services and determine if a service is relevant according to user's query. The discovery in this phase is to do the matchmaking between the functional requirements and the WSDL descriptions of web services. After the functional matchmaking, there is a high probability that the user gets an extensive list of web services which fulfill the functional requirements of user's query.

To filter through this list of functionally similar services, users must be further assisted in selecting the appropriate web service from the list, which is the second phase, the selection phase. -In this phase, the user will filter the list based on his non-functional requirements. In this work, for non-functional requirements, we mainly concentrate on the Quality of Service (QoS) attributes. QoS can be used for discriminating between functionally equivalent web services. For example, there is a user who is looking for a web service whose availability is greater than 90% and throughput is more than 34.2 invokes per second and documentation percentage is bigger than 85%. There may be many services satisfying user's functional requirements, but only a few satisfying all these non-functional requirements. Then it will be easier for the user to decide which service to choose.

1.1.2 Problem Statement

Automatic service selection, especially QoS-based service selection, has been an active research area in recent years. Various models such as Multi-Criteria Decision Making (MCDM) [1], Constraint Programming (CP) and Mixed Integer Programming (MIP) [2], Skyline [3], have been used to process users' requirements (i.e., constraints and preferences) on multiple QoS criteria, in order to find services that could optimize those criteria. Normally the optimization is

considered to be achieved if a service has the best overall QoS value among all alternative services. For instance, a commonly used metric is the weighted sum of multiple QoS values of a service.

In reality, this default optimization strategy may not always work. According to the decision making theory [4], there are many different strategies people may follow when they decide what to choose among a list of alternatives based on multiple criteria. Definitely, different strategies would result in different selection results. Consider an example of selecting a web service, in which users make their decisions based on three QoS properties – price, reliability, and rating. Suppose QoS values of Service 1 (*S1*) on these three properties are ($50, 95%, 3.5 stars), QoS values of Service 2 (*S2*) are ($85, 99%, 5 stars), and QoS values of Service 3 (*S3*) are ($50, 75%, 4 stars). And suppose there are three users *A*, *B*, and *C*, all of them have the same selection criteria: (price < $85, reliability > 85%, rating > 3), and all the criteria are negotiable. When deciding which service to select, User *A* follows a strategy in which he checks the most important criterion to him first, which is price, after finding out which services (*S1*, *S3*) are optimal on this criterion, he moves on to check the next important criterion, which is rating, and then he selects *S3*. User *B* follows a different strategy in which he checks which service wins on the most number of criteria (*S1* wins on one QoS criteria – price, *S2* wins on two – reliability and rating, *S3* wins on one – price), and since *S2* is a clear winner and *S2* is selected. User *C* follows yet another strategy in which he checks the sum of all three QoS values, since *S1* and *S3* gain a lot on price compared to *S2*, and *S1* leads *S3* on reliability more than *S3* leads *S1* on rating, the overall value of *S1* is the best, and thus he selects *S1*. From this example, we could see that even with the same QoS requirements, when users follow different decision strategies, different services are selected.

Furthermore, one user may follow different strategies in different contexts or for different tasks. For instance, when users select free services for leisure purposes, usually compensatory strategy [5] such as weighted sum on multiple values would work because they do not mind the trade-off on some criteria if necessary. However, when the same users select services for their work, non-compensatory strategy [5] such as the one User *A* follows in our previous example would be a better option because selection is more serious in this context, some criteria simply cannot be compromised, and preference could play a bigger role.

From these analyses, we could see the clear gap between the way current selection system works and the real scenario when people may follow a variety of decision strategies during the selection process. Our objective in this work is to fill this gap and make the automatic selection system closer to the way users actually follow when selecting services manually. Our system would help users select web services based on not only their QoS requirements but also their preferred decision strategies.

Sometimes it could be hard for users to clearly and unambiguously define strategies they follow as decision making is often a subconscious and subtle process. Also the strategy may change over the time or in different context or depending on purposes of services. To take all these into consideration, we apply the learning to rank technique [6] on the historical data to identify decision strategies users followed in the past. The goal is to personalize the service selection algorithm for individual users. We assume that the history data on users' service selection patterns are available through server logs saved in the repository or through other means, so that a personalized ranking model can be learned. Also to simplify our problem, we only consider the single service selection, not in the context of service composition.

1.2 Contributions

There are two major contributions of the work: 1) considering that user's decision strategy plays an important role in the service selection process and ignoring it will affect the selection accuracy, we have implemented different user decision strategies with the combination of different rules in QoS-based service selection approach in which both QoS criteria and decision strategies are taken into account; 2) since users may follow multiple decision strategies depending on the context and in an implicit way, we propose to apply the learning to rank technique to find the best matching strategies and the best ranking model combining them. The proposed approach is flexible and extensible so that we can plug-in different QoS-based selection models, decision strategies, as well as learning to rank algorithms.

1.3 Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2 reviews and analyzes the existing research work in the field of web service selection. Then, the research efforts that are more closely related to our work such as QoS based web service selection, personalized service ranking, learning to rank and user decision strategies are reviewed.

Chapter 3 gives the details of our proposed approach, including the service selection and ranking algorithms based on different decision strategies. We are considering both the QoS criteria and decision strategy for the service selection. We have implemented four decision strategies and 3 rules for the service selection. Chapter 3 also explains the architecture model of our selection system, and the learning process to obtain the personalized service ranking model. We apply the learning to rank algorithm for finding the best matching decision strategies and the best ranking model for the service selection.

Chapter 4 explains our experiment design, shows the simulation process to generate the dataset, and analyzes and discusses the results.

Finally, in Chapter 5, we conclude our thesis with a summary of results and analysis. Our future research directions are also discussed in this chapter.

CHAPTER 2

RELATED WORKS

2.1. Introduction

In the last decade web is changing from a web of pages to a web of services. So instead of maintaining the static documents the web is giving more and more access to web services. Web services can be easily integrated both into the traditional software and into the web pages. In today's distributed environment web services are becoming an important part of business applications. It is becoming more and more popular for implementing the business solutions. Because of the large number of services it is very difficult to select among a list of services. To decide which service is the best with all their functional and non-functional requirements of the users is a decision problem. It is essential to analyze properly before the selection of web services as it involves multiple criteria and interdependent relationship between them. Service selection is divided into two steps: match the services based on the functional requirements where functional requirements determine the overall behavior of web services and then select and rank the services based on non-functional requirements which are not directly related to the functionality of web services. Typically non-functional requirements are requirements on the Quality of Service (QoS) attributes offered by web services such as response time, reliability, availability, scalability, reputation etc. QoS is a main element that differentiates the web services that have the similar functionalities. QoS based service selection involves different steps that are alignment, matchmaking, optimization and selection. Alignment is responsible for aligning the QoS metrics of QoS offers and demands during the web service publishing stage. Matchmaking is responsible for matching the QoS offers and demands. Optimization step comes in if there is

no matching result during the matchmaking step. In the end there is a selection step that is responsible for ordering the matchmaking results based on the weights provided by the customer.

We will first review various QoS-based service selection models proposed in the past researches. Then, we will review some of the literatures that are more closely related to our work.

2.2. QoS-Based Web Service Selection

QoS-based service selection is usually considered as an optimization problem, and thus various optimization models have been used to solve this problem. Their main target is to help achieve the quality tradeoffs and optimizations.

In [7], users define their preferences using utility functions, quality distributions of providers are learned from a probabilistic trust model, and then the services which could maximize the expected utility are selected.

In [2], Mixed Integer Programming (MIP) is used for solving the problem of web service selection and match-making. Here mixed integer programming is used for solving the linear constraints while Constraint Programming (CP) is used for non-linear constraints. The experiments conducted in this paper showed that MIP outperforms the CP while considering the linear constraints. So the process of match-making is completed using CP or MIP depending upon the type of constraints whether it is non-linear constraints or linear constraints.

In [1], WS-QoSOnto is proposed for designing and building QoS ontology for web services. This proposed ontology has many aspects for describing the QoS properties, metrics, trends and relationships. For defining the user preferences an Analytic Hierarchy Process (AHP) model is used which is a multi-criteria decision making approach for developing a flexible and dynamic ranking algorithm. Pair-wise comparison is the main step in AHP which determines

how a particular QoS property is more important than the other QoS property in terms of ranking the web services. Given a service request, an AHP hierarchy is generated first, and then weights are calculated for QoS properties and final ranking scores are the weighted sum of all the QoS values. Eigen vector and Eigen values are used for calculating the relative weights of the QoS properties. The main problem in this proposed model is that while calculating the overall service ranking the proposed model completely ignores the user defined QoS requirements. It does not include the user defined constraints over the QoS properties.

Skyline computation is used for solving the problem of QoS based web service selection in [3]. In this paper the authors are dealing with the issues that the users are required to convert personal preferences into numeric weights and the users may not have enough knowledge to convert these preferences into numeric weights. And the second issue discussed in skyline approach is how to deal with the dynamic environment of the QoS properties. To deal with these issues, p-dominant skyline is used for the service optimization problem with dynamic QoS information and in p-dominant skyline it is not mandatory for the users to provide the preferences for the QoS properties. It finds optimized solutions by measuring their dominance relationships. P-dominant service skyline is a tool that sets the threshold and determines whether the service provider should come into the preferred services list. A service provider *S* belongs to the p-dominant service skyline if the chance that *S* is dominated by any other service provider is less than *p,* where *p* is the probability threshold. Efficiency and quality issues are further addressed in [8] by using a skyline variant – σ-dominant skyline.

Most of the above selection systems did not consider the variety of the individual decision strategies involved in the selection process whereas we do.

2.3 Personalized Service Ranking

Personalized service ranking and selection has attracted research attention in recent years. Most of the efforts in this area are using recommendation algorithms. In [9], personal profiles are built which is based on the collaborative filtering technique. Collaborative filtering technique has been extensively used in recommender systems where the consumers rate the web services. And then based on the previous user ratings the system does the predictions for the services which the consumers have not used before or rated before. System finds the similar users based on their invocation histories. Collaborative filtering technique has some limitations as it is based on the explicit ratings from the consumers, which sometimes is not possible in real life because it is not necessary that every consumer will rate the service. To overcome this limitation, the work in [9] is also considering the dependency among the services. In a real world it is quite possible that one web service does not fulfill the requirements of consumers. So there is a need of joint service compositions for fulfilling the consumer's requirements. The paper used the association rule mining to identify service dependencies based on the past composition transactions of similar users. After applying the collaborative filtering technique and considering the dependency among the web services, the web service usage history is stored in the consumer's personal profile. Next time when the consumer logged in this personal profile will be helpful for ranking the web services.

In [10], user similarity is measured as the similarity between the rankings of their observed QoS values on commonly invoked services, and the QoS driven component ranking framework for cloud applications is implemented using the past experiences from similar users.

For personalization it is necessary to evaluate all the components at user side and rank the components based on the QoS values. But it is difficult to do this in cloud applications, because

it is a resource consuming and time consuming task and it is also not possible for providers to evaluate all the cloud components since there could be a large number of components in the cloud. To overcome these issues a collaborative QoS driven quality ranking framework is implemented which measures the user similarity with the current user based on the ranking order on the commonly invoked components. Based on the user similarity a set of similar users will be identified. In the next step a preference function is modeled that measure the quality priority of the components. In the last an algorithm is implemented that ranks the components based on the user invocation histories and preference functions.

In [11], previous interactions between service providers and requestors are modeled as a social network. Information is collected using the data mining techniques on the log that contains the information about the past communications and requests of web services by different consumers. It finds out the likely social link between service providers and service consumers. Ranking of the web services is done from three different aspects, the consumer aspect, the service aspect and the non-functional properties aspect. And then relevancy is calculated of different web services for different consumers. The work in [11] also discusses the scenario when they have the partial information from the above mentioned three aspects. In that case social network analysis and graph theory techniques are applied for the calculation of relevancy of the web services. At the end results from the social network analysis are fed into a Bayesian classifier to rank services.

In [12], invocation and query logs are used to calculate user similarities and then services are ranked based on most similar users' invocation histories. After functionally matching services are found out, they are ranked based on the current QoS requirement and how they were selected and invoked by past users with the same QoS requirements. Invocation and query logs could

show how and when the users submitted the service request to the search engine. When two users invoke the same service they are considered to be similar and in addition to that if both users have the similar QoS requirements then their similarity level will be high because it shows that both users are following the same decision making patterns.

In [13], both functional interests and QoS preferences of users from their usage history are considered for similarity calculation and service recommendation. Here the usage history comprises of previous web services and their matching preference records and these records are based on WSDL documents of web services invoked by the consumer and the consumer's preference records. The proposed framework also calculates the similarity between the consumer's functional requirements and web services. Then hybrid approach is developed that merges both the functional similarity and non-functional similarity. At the end top *k* web service list is generated for the consumer that considers both the functional and non-functional requirements from the usage history.

Compared to these algorithms, although we also use the past query and invocation histories, we tackle this problem from a totally different perspective. We consider that users may follow different decision strategies for different queries, and we use learning to rank algorithm [6] to learn a personalized decision-strategy-based ranking model.

2.4 Learning to Rank

Learning to rank has been widely used in information retrieval and web searching for ranking the documents. And its main purpose is to automatically construct a ranking model using training data for ranking objects / entities. Learning to rank techniques can be divided into three types, which are point-wise, pair-wise and list-wise techniques and these techniques are based on the types of input objects for learning. In Point-wise technique input object is a single document

and it requires a relevance judgment for each document in the training set, whereas in pair-wise technique input object is a pair of documents and it requires a relevance judgment for each pair of documents and in list-wise technique input object is a list of documents and it requires a relevance judgment for a list of documents. Point-wise and pair-wise techniques do not consider the interdependency among documents and hence position information is missing in these documents, while list-wise technique considers the position information in the resulting ranked list. It has been proved in [6] that list-wise ranking algorithms give the better performance than the point-wise and pair-wise algorithms.

Freund et al. [24] developed a learning to rank algorithm called Rank Boost and applied it on information retrieval problem. Rank Boost is a pair-wise boosting algorithm that is based on AdaBoost algorithm. It learns a ranking function by combining many weak rankings of the given instances. Each weak ranker may have only a weak correlation to the target ranking. Weak ranker can have any type but most of the time it chooses the binary function with a single feature. Here the algorithm starts in rounds where it assigns the equal weights to the pairs of documents. At each round the algorithm chooses the weak ranker that has the lowest pair-wise loss on the training data as per the current weight distribution. Weights will be decreased for those pairs which are correctly ranked and will be increased for those pairs which are incorrectly ranked. So the algorithm can put more focus on those pairs which are incorrectly ranked. At the end the output is the single highly accurate ranking.

Burges et al. [23] introduced the RankNet algorithm and showed the effectiveness to improve the search relevance. RankNet requires a label dataset for training the model. The method requires the pair-wise preference information together with the gradient descent for

training the model. The algorithm is simple to use and provides good performance on a large amount of data.

Cao et al. [25] developed a list-wise approach named ListNet for learning to rank and did the comparison with the pair-wise approaches such as RankNet, Rank-Boost, Ranking SVM. They established that list-wise approach is more effective as compare to pair-wise approaches. Problem with pair-wise approach is that object pairs fluctuates largely from query to query and this could lead to bias in which the preference will be given to those queries which have more object pairs. In list-wise approach lists of objects are used as instances whereas pair-wise approaches use object pairs as instances. In list-wise approach probabilistic method has been used for the calculation of list-wise loss function and they are using permutation probability and top one probability for the conversion of ranking scores into probability distributions. ListNet is using the Neural Network and Gradient Descent for the development of learning method.

Metzler and Croft [26] discussed the linear feature based models in detail. Here the model's scoring function consists of the linear combination of features which combines the multiple sources of ranking evidence in document retrieval. Most of the features which are used in the models include term occurrence / non-occurrence, term frequency, inverse document frequency, document length, term proximity and these are called primitive textual features. The models directly optimize the retrieval metric of choice. It has been shown that feature based models considerably outperforms current state of the art retrieval models with the right choice of features.

2.5 User Decision Strategies

There are many user decision strategies people may follow when they make decisions like which product or service they need to choose and which product or service they should

ignore; which brand to select and which brand to ignore. Different types of strategies have been introduced for solving these problems in which consumers are having difficulties of selecting the services or products among the large number of services / products with multiple attributes.

In [5], two types of strategies are discussed, compensatory and non-compensatory. In compensatory strategies all the information is processed and tradeoff is made between the high value on one attribute of an alternative and a low value on another attribute. In these types of strategies one attribute is losing the quality in return of gaining the quality of other attributes. Two commonly used compensatory strategies are weighted additive (WADD) and equal weight (EQW). WADD strategy examines all the available information. In WADD strategy decision maker gets the relative importance of each criterion by placing weight on each criterion. EQW strategy examines all the available information but it ignores the weight information. It assigns the equal weight to all attributes. Non-compensatory strategies do not make use of all the available information and trade-off is normally ignored. Here the presence of one attribute may not be able to compensate the absence of other attributes. These strategies are used to reduce the number of attributes to be sensibly evaluated. Two commonly used non-compensatory strategies are elimination by aspects (EBA) and lexicographic (LEX). EBA strategy starts by finding the most important attribute (the one which has the highest weight). Then the cut off value of each attribute is retrieved, and all the alternatives whose values are less than the cut off values are removed. This process continues until one alternative is left. In LEX strategy all alternatives are ranked based on the most important attribute. If there are two alternatives whose attribute value is similar then the second most attribute value will be considered; this process continues until one alternative is left.

In [14], more strategies are discussed and they are classified based on their characteristics, such as whether all attribute values are processed or some of the attribute values are processed, whether they are attribute based or option based. Attribute based processing consider all the options of a single attribute while option based processing consider all the attribute values of a single option before considering the next option. Some other characteristics include whether they are eliminating the options before making the final decision, whether same amount of information is examined or it changes across attributes, whether decision strategies are considering the weight of each attribute or not, whether the decision strategy uses the cut off levels for the attributes or not, and whether the decision strategy use the quantitative reasoning or qualitative reasoning. Quantitative reasoning involves the addition, subtraction and multiplication of attribute values whereas qualitative reasoning involves the comparison of attribute values. The paper also explains the classification method that is used for identifying the user decision strategies. And this method comprises of four metrics and these metrics are divided into process based and outcome based.

According to [15], decision makers may use multiple strategies for making choices and then select strategies from a range of strategies that represent the best accuracy and choice for the particular decision problem. The selection of the strategy is depending on the task and context factors. Task factors represent the basic features of a decision problem like response mode and number of alternatives available. Context factors represent similarity of alternatives which check the similarities of alternatives features. When less number of alternatives is involved consumers often use those strategies that examine all the available information like WADD, EQW and trade-off is made between the attributes. When large number of alternatives is involved consumers

often use heuristic strategies like EBA and cut off value is introduced for cutting down the number of alternatives.

The variety of the individual decision strategies is also recognized in some domain-specific applications, and integrating multiple strategies is proved to be able to provide a better result [16]. Compensatory and non-compensatory aggregation strategies have been used in multi criteria analysis for decision making. In [16] two aggregation strategies are combined with a single weighting strategy. The strategies which were combined are weighted summation, Electre II aggregation strategy and the Rank Order Centroid (ROC) weighting strategy where weighted summation is a compensatory strategy and Electre II is a non-compensatory strategy.

Compared to these papers, we do not require users to define their strategies explicitly, we generate the ranking order of all alternative services instead of just finding the best one, and we define an automated solution by using a learning to rank algorithm to find the optimal way to combine multiple strategies.

CHAPTER 3

PERSONALIZED SERVICE RANKING BASED ON DECISION STRATEGIES

In this work, we assume that the functionally matched web services have been selected [28]. This search can be done on registries or using a P2P discovery mechanism [29]. Here registries are the main repository for the web services where service providers publish and store their web services and later on these services will be invoked by service consumers. In a P2P discovery mechanism web services are the nodes in the network. The requested web service will be matched or the request will be spread through the network for finding the suitable web service.

The next step is to select the web services based on the non-functional requirements. Here the main task is to find the list of optimal web services based on the user requirement on various QoS properties. In this chapter, we first explain our weighting scheme, the QoS properties which we have used in our selection process as well as the decision strategies which we consider in the service selection process for selecting and ranking the web services. Then we show a few scenarios where a user may follow different strategies. After that, we describe our system architecture, especially on how we collect the historical data. Finally we discuss how we use a learning-to-rank algorithm – AdaRank [17], to learn our personalized service selection algorithm.

3.1. Weighting Scheme

The weights of the QoS properties represent the importance level for the users while making the trade-off decisions.

In our selection system the system displays the list of available QoS properties and the user is required to select the preferences / weights over QoS properties of web services. The user can provide the preferences to as many QoS properties as the user wants. For weighting the QoS criteria we have used the simple weighting scheme that calculates the direct importance of the QoS properties. For weighting, our selection system provides the scaling scheme that ranges from 1 to 9; where 1 is the least important and 9 is the most important. Saaty [32] proposed a scale of relative measurement that we are using for assigning the preferences to QoS properties.

Table 3.1: Fundamental scale of numbers indicating the importance level of QoS properties [32]

Importance Level	1	2	3	4	5	6	7	8	9
	Least important	Weak important	Moderate important	Moderate plus	Strong important	Strong plus	Very strong	Very, very strong	Most important

If no weight is assigned to the QoS property then zero weight will be assigned to that particular QoS property which shows that the user has no preference over this particular QoS property. The advantage of our selection system is that we are giving the flexibility to the users in assigning the QoS weights and users are not required to give the excessive information.

Next step is to normalize the weight so the sum of each weight vector is 1. For normalizing we have used this formula:

$$w_i = \frac{q_i}{\sum_{j=1}^{k} q_j} \qquad \textbf{(3-1)}$$

In the above equation w_i is the normalized weight of the i-th criteria, q_i is representing the user defined weight of the i-th criteria and k is representing the total number of criteria the user selected.

3.2. Constraints on QoS Properties

In a user's QoS requirement, there could be constraints on QoS properties. For example the user can request that reliability > 90%, availability > 98%, successability > 80%, which means services should be checked first on whether they satisfy these constraints and then on whether they have the optimal values on these properties. In the selection and ranking of web services these constraints play an important role in determining a list of optimal services and their ranking orders.

3.3. Quality of Service (Non-Functional) Properties of Web Services

As we are dealing with the Quality of Service based web service selection problem we need to identify the QoS criteria [30] [31] which are important for web service selection. A web service can have many non-functional properties such as:

- **Availability:** measures the probability that the system is up and can be accessed by users successfully. It calculates the number of successful invocations / total number of invocations. It is measured in (%).
- **Reliability:** measures the ability of a system to work as expected under specific state for a specific period of time. It calculates the ratio of number of error messages to total messages. It is measured in (%).
- **Throughput:** calculates the volume of data which is invoked at a given period of time. It is measured as (invokes/sec).

- **Response Time:** is the time taken from sending a request to receiving a response. It is measured in (ms).
- **Successability:** measures the requests that have been successfully completed. It calculates the total number of responses / total number of requests. It is measured in (%).
- **Compliance:** measures to what extent a WSDL document follows WSDL specifications. It is measured in (%).
- **Best Practices:** measures to what extent a web service follows WS-I Basic Profile. It is measured in (%).
- **Cost:** is the cost which the customer needs to pay for using the service.
- **Documentation:** measures to what extent the documentation is completed like description tags in WSDL. It is measured in (%).

In our selection system we are using 7 QoS properties including Availability, Throughput, Successability, Reliability, Compliance, Best Practice, and Documentation. Our selection system is flexible and we can add more QoS properties if we want. But for the simplicity currently we are using the 7 QoS properties in our system.

3.4. Service Selection Based on Decision Strategies

The next phase in our service selection system is selecting and ranking the web services. This selection and ranking is based on the QoS weights, user constraints and requirements on the QoS properties which are already defined in the previous sections. For the selection and ranking of web services we are using the decision strategies which will be explained in this section.

There are many decision strategies reported in the literature [5] [14]. Here, we mainly consider four of them, namely, Weighted Additive (WADD) strategy, Majority of Confirming

Dimensions (MCD) strategy, Weighted Majority of Confirming Dimensions (WMCD) and Lexicographic (LEX) strategy [5]. These strategies are used to reduce the number of alternatives and improve the processing efficiency.

Since most of the decision strategies only target at finding optimal solutions without handling the constraints, we develop two rules which could rank services based on how well they satisfy the constraints. We also modify the process of the original decision strategy in order to generate a full ranking order on services. Below, we give definitions and explanations on the two rules and the three decision strategies.

- **Layer Rule:** Services satisfying all of the constraints (category 1) are ranked higher than services satisfying some of the constraints (category 2), which are ranked higher than services satisfying none of the constraints (category 3).
- **Quantity Rule:** A service satisfying more constraints is ranked higher. To simplify the case, we do not consider the user preferences on QoS criteria, and only count the number of satisfying criteria.
- **WADD Strategy:** A service is ranked higher if the sum of its QoS values multiplied by their corresponding weights is higher. Based on this strategy, a ranking score of a service s_i is calculated as $\sum_{k=1}^{M} w_k.a_{ik}$, where k represents the k-th QoS property, M is the number of properties the system supports, w_k measures its weight, and a_{ik} represents the value of s_i on the k-th property. This is the ranking algorithm used by many service selection systems [1] [2] [3].

  Implementation of WADD strategy:

  Suppose the user has selected 3 criteria / QoS properties, e.g. criterion1, criterion2, criterion3

Step1:

In step1 we will normalize the weight of each criterion by using Equation 3-1.

Normalized Weight of Criterion1 = (Criterion1 individual weight) / (Total weight of all the criteria)

Normalized Weight of Criterion2 = (Criterion2 individual weight) / (Total weight of all the criteria)

Normalized Weight of Criterion3 = (Criterion3 individual weight) / (Total weight of all the criteria)

Step2:

Normalize the web service values for each criterion in the database.

Select ServiceName, (Criterion1 / (select max(Criterion1) from QWSDataset)), (Criterion2 / (select max(Criterion2) from QWSDataset)), (Criterion3 / (select max(Criterion3) from QWSDataset)) from QWSDataset

Step3:

For WADD implementation multiply the normalized values of each service with the respected values of each criterion weight. And then sum up all the criteria values for each service.

$$Score_j = \sum_{k=1}^{M} w_k.a_{ik} \qquad \textbf{(3-2)}$$

As we are calculating the score based on 3 criteria so $M = 3$ and w_k is representing the weight of the k-th criterion calculated in step1.

Step4:

Rank all the services based on the score calculated in step3. The ranking list will be in descending order.

- **MCD Strategy:** A service with a majority of winning (better) QoS values is ranked higher when compared with another service. This strategy involves processing pairs of services. The values for each of the two services are matched on each QoS property and the service with a better property value will be selected. In case of tie between two services our application will select the service which has the better value in the last property. And then this selected service will be compared with next service among the set of services. The comparison stops when all the services have been compared and the final winning service has been selected. We do the comparison on all pairs of services to get the complete ranking order of these services.

  A variation of this strategy is to include weights in the ranking process. When comparing two services s_i and s_j, if s_i is better than s_j on the k-th QoS property, p_{ijk} is set to 1, otherwise is 0. Based on the original MCD strategy, if $\sum_{k=1}^{M} p_{ijk} > \sum_{k=1}^{M} p_{jik}$, s_i is ranked higher than s_j. Based on the weighted MCD (WMCD) strategy, if $\sum_{k=1}^{M} w_k . p_{ijk} > \sum_{k=1}^{M} w_k . p_{jik}$, s_i is ranked higher than s_j.

Implementation of MCD strategy:

Suppose the user has selected 3 criteria / QoS properties e.g. criterion1, criterion2, criterion3. For MCD implementation we need to compare the value of each service on each category.

Step1:

Compare the value of s_i with s_j for each category. If s_i has better value than s_j for one category then assign 1 otherwise assign 0. Add up all 1's for each service. The service which has more 1's or the highest number will be the winning service.

$$\sum_{k=1}^{M} p_{ijk} > \sum_{k=1}^{M} p_{jik}, \qquad \textbf{(3-3)}$$

Step2:

Repeat step1 for all the services until we have the winning service for all pairs and we get the complete ranking list of all the services.

- **LEX Strategy**: The services are first ranked on the most important property (i.e., with the highest weight), if there are ties, services that have the same value on this property are then ranked on the second most important property, and this process continues until a full ranking order on all services is generated.

  There is another variation of this strategy that is called lexicographic semi order. This strategy is similar to lexicographic strategy but it also introduces the idea of Just Noticeable Difference (JND). If there are some services which comes under JND of the best service on the most important QoS property they are considered to be equal. The reason to introduce this term is because the services which are marginally better on the most important property will be selected but much worse on the other properties will be ignored. Due to the time constraint, although we have implemented this strategy, we didn't run the experiment using this strategy. Therefore, in the rest of the discussion, this strategy is not included.

Implementation of Lexicographic strategy:

Suppose the user has selected 3 criteria / QoS properties e.g. criterion1, criterion2, criterion3. For LEX implementation we will select the criterion which has the highest weight. For example criterion1 has the weight of 7, criterion2 has the weight of 5, criterion3 has the weight of 9, and then our selection query will be like this:

select ServiceName, criterion3, criterion1, criterion2 from QWSDataset order by criterion3 desc , criterion1 desc , criterion2 desc

Services are ranked based on criterion3, if there is a tie then it will rank the services based on criterion2, this process continues until we get the complete ranking order.

In our system, the service ranking algorithm is based on one of the combinations between rules and strategies. The three rules (No Rule, Layer Rule, and Quantity Rule) decide how a user wants the system to handle the constraints, and the four decision strategies (WADD, MCD, WMCD, and LEX) define the preference-guided optimization process. If the number of satisfying constraints really matters to a user, Quantity Rule is applied in the ranking process. If the number of satisfying constraints is important but the exact number is not so critical, Layer Rule is applied. If a user does not care about satisfying constraints, No Rule is applied. If no rule is applicable, one of the four strategies is used directly to rank all the services. If Layer Rule is selected, services are first ranked based on the Layer Rule, and then services which fall into the same category are ranked according to one of the four strategies. If Quantity Rule is selected, services are first ranked based on the Quantity Rule, and then services which satisfy the same number of constraints are ranked according to one of the four strategies. There are in total 12 combinations, and thus we have 12 decision-strategy-based ranking algorithms.

3.5. Detailed Explanation of Service Selection System (Case Study)

In this section we will provide a detailed case study that explains the weighting, and the selection and ranking process of our QoS based and decision strategy based web service selection framework.

3.5.1 General Settings

For our framework we are using the real time dataset - the QWS Dataset [22] which has 2,507 web services. We have implemented the framework on all 2,507 web service. But for illustration purpose currently we are using seven web services and six QoS properties. Table 3.2

has the complete information which is needed for our illustrating example. In the table we have included the QoS criteria / properties, QoS Tendency, Units used for QoS properties, list of services and their values, user request on each QoS property. QoS tendency has two values: high and low. In case of high value the service which has high value on a particular QoS property will be preferred and in case of low value the service which has low value on a particular QoS property will be preferred. For simplicity we are using the high values only for QoS properties. QoS units are used for measuring the QoS metrics and most of the QoS properties have % unit in our case study.

Table 3.2: List of functionally matching services

QoS Criteria	AV	SS	RL	CL	BP	Doc
QoS Tendency	High	High	High	High	High	High
Unit	%	%	%	%	%	%
User Request	>88	>=96	>70	>80	>=82	>=60
MAPPMatching	89	90	73	78	80	32
Compound2	85	95	73	100	84	2
USDAData	89	96	73	78	80	96
GBNIRHolidayDates	98	100	67	78	82	89
CasUsers	87	95	73	89	62	93

Here AV stands for Availability, SS stands for Successability, RL stands for Reliability, CL stands for Compliance, BP stands for BestPractices, Doc stands for documentation. These QoS properties have already been defined in Section 3.3. MAPPMatching, Compound2, USDAData, GBNIRHolidayDates and CasUsers are service names.

3.5.2 Calculating the Default Weight of QoS Properties

In this section we will explain how we calculate the default weights for QoS properties. There are many weighting methods which can be used for the calculation of weights. But for simplicity we are using the simple weighting scheme only to generate the QoS weights for the

user preferences. Suppose the user has submitted the preferences for the QoS criteria through our selection framework. The user preferences are given in Table 3.3.

Table 3.3: User preferences on QoS properties

QoS Criteria	AV	SS	RL	CL	BP	Doc
Weights	8	8	8	6	5	5

Based on the above table, availability, successability and reliability are the most important criteria for the user followed by throughput, compliance, best practices and documentation. In the next step we normalize these user defined weights. For normalizing the user defined weights we are using the formula (3-1). Table 3.4 shows the normalized user defined weights

To apply this formula, first of all we calculate the total weight for all the criteria.

Total weight = 8 + 8 + 8 + 6 + 5 + 5 = 40

Table 3.4: Normalized user defined weights

QoS Criterion	AV	SS	RL	CL	BP	Doc
Normalized Weights	0.2	0.2	0.2	0.15	0.13	0.13

3.5.3 Selecting and Ranking the Web Services Using Decision Strategies

3.5.3.1. Using WADD Strategy:

After the calculation of QoS weights, to continue with the ranking process the next step is to normalize the web services' QoS values. For normalizing these values we have used the idealizing approach [38]. The idealizing approach is applied on each criterion by assigning 1 to the web service with the highest value and then divides each service value with the highest one. This process is done for all QoS criteria.

Next step is to apply the weighted sum method for getting the ranking score. This is done by multiplying the weight vector with the normalized values of QoS criteria. Then at the end add up all the QoS values of each service for getting the final ranking score. The calculation of web service scores using the WADD strategy is shown below.

Table 3.5: Calculation of web services score using WADD Strategy

QoS Criterion	**AV**	**SS**	**RL**	**CL**	**BP**	**Doc**
MAPPMatching	0.91 * 0.2 = 0.18	0.9 * 0.2 = 0.18	1 * 0.2 = 0.2	0.78 * 0.15 = 0.12	0.95 * 0.13 = 0.12	0.33 * 0.13 = 0.04
Compound2	0.87 * 0.2 = 0.17	0.95 * 0.2 = 0.19	1 * 0.2 = 0.2	1 * 0.15 = 0.15	1 * 0.13 = 0.13	0.02 * 0.13 = 0.003
USDAData	0.91* 0.2 = 0.18	0.96 * 0.2 = 0.19	1 * 0.2 = 0.2	0.78 * 0.15 = 0.12	0.95 * 0.13 = 0.12	1 * 0.13 = 0.13
GBNIRHoliday Dates	1 * 0.2 = 0.2	1 * 0.2 = 0.2	0.92 * 0.2 = 0.18	0.78 * 0.15 = 0.12	0.98 * 0.13 = 0.13	0.93 * 0.13 = 0.12
CasUsers	0.89 * 0.2 = 0.18	0.95 * 0.2 = 0.19	1 * 0.2 = 0.2	0.89 * 0.15 = 0.13	0.74 * 0.13 = 0.1	0.97 * 0.13 = 0.13

MAPPMatching = 0.18 + 0.18 +0.2 + 0.12 +0.12 +0.04 = 0.84

Compound2 = 0.17 + 0.19 + 0.2 +0.15 +0.13 + 0.003 = 0.843

USDAData = 0.18 + 0.19 + 0.2 + 0.12 + 0.12 +0.13 = 0.94

GBNIRHolidayDates = 0.2 + 0.2 + 0.18 + 0.12 + 0.13 + 0.12 = 0.95

CasUsers = 0.18 + 0.19 + 0.2 + 0.13 + 0.1 + 0.13 = 0.93

Table 3.6: Final Ranking order based on WADD strategy

Ranking Order	Web Services	Total Score
1st	GBNIRHolidayDates	0.95
2nd	USDAData	0.94
3rd	CasUsers	0.93
4th	Compound2	0.843
5th	MAPPMatching	0.84

3.5.3.2. Using MCD Strategy:

MCD strategy requires a pair-wise comparison. QoS values of two services are compared and the service which has a higher number of winning QoS values will be selected and then the winning service will be compared with the next service. If service1 has the better QoS value over service2 then it will be set to 1 otherwise it will be set to 0. We start by doing the comparison to the first pair of web services listed in Table 3.2.

Table 3.7: Step1 – Comparison of web services in MCD strategy

QoS Criterion	**AV**	**SS**	**RL**	**CL**	**BP**	**Doc**
MAPPMatching	1	0	0	0	0	1
Compound2	0	1	0	1	1	0

In the above table, MAPPMatching service is better on 2 QoS values so it has 2 1's and Compound2 service is better on 3 QoS values so it has 3 1's. As Compound2 service has more 1's than MAPPMatching service, Compound2 service is the winning service and is compared with the next pair.

Table 3.8: Step2 – Comparison of web services in MCD strategy

QoS Criterion	**AV**	**SS**	**RL**	**CL**	**BP**	**Doc**
Compound2	0	0	0	1	1	0
USDAData	1	1	0	0	0	1

In the above table USDAData is the winning service and is compared with the next service.

Table 3.9: Step3 – Comparison of web services in MCD strategy

QoS Criterion	**AV**	**SS**	**RL**	**CL**	**BP**	**Doc**
USDAData	0	0	1	0	0	1
GBNIRHolidayDates	1	1	0	0	1	0

In the above table GBNIRHolidayDates is the winning service and is compared with the next service.

Table 3.10: Step4 – Comparison of web services in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
GBNIRHolidayDates	1	1	0	0	1	0
CasUsers	0	0	1	1	0	1

In the above table both services have the same number of winning attributes. GBNIRHolidayDates and CasUsers web services both have 3 winning attributes or 3 1's. So if there is a tie then we will consider the winning service to that service which has a better value on the last attribute. In our case CasUsers will be the winning service and it will be ranked # 1 among the list of services. In the next phase we will remove this winning service from the list of services and will do the comparison for the rest of services.

Table 3.11: Remaining list of web service in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
MAPPMatching	89	90	73	78	80	32
Compound2	85	95	73	100	84	2
USDAData	89	96	73	78	80	96
GBNIRHolidayDates	98	100	67	78	82	89

Again the comparison will be done for the rest of the services.

Table 3.12: Step5 – Comparison of web services in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
MAPPMatching	1	0	0	0	0	1
Compound2	0	1	0	1	1	0

Table 3.13: Step6 – Comparison of web services in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
Compound2	0	0	0	1	1	0
USDAData	1	1	0	0	0	1

Table 3.14: Step7 – Comparison of web services in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
USDAData	0	0	1	0	0	1
GBNIRHolidayDates	1	1	0	0	1	0

In these comparisons winning service is GBNIRHolidayDates and it will be ranked # 2 among the list of services and it will also be removed from the list of web services. Now we will do the comparison to the rest of web services.

Table 3.15: Remaining list of web service in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
MAPPMatching	89	90	73	78	80	32
Compound2	85	95	73	100	84	2
USDAData	89	96	73	78	80	96

Again the comparison will be done for the rest of the services.

Table 3.16: Step8 – Comparison of web services in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
MAPPMatching	1	0	0	0	0	1
Compound2	0	1	0	1	1	0

Table 3.17: Step9 – Comparison of web services in MCD strategy

QoS Criterion	AV	SS	RL	CL	BP	Doc
Compound2	0	0	0	1	1	0
USDAData	1	1	0	0	0	1

In these comparisons winning service is USDAData which will be ranked # 3 and now this winning service will be removed from the list of web services. The comparison will be done to the rest of web services.

Table 3.18: Remaining list of web service in MCD strategy

QoS Criterion	**AV**	**SS**	**RL**	**CL**	**BP**	**Doc**
MAPPMatching	89	90	73	78	80	32
Compound2	85	95	73	100	84	2

Now the comparison will be done to the last set of pairs.

Table 3.19: Step10 – Comparison of web services in MCD strategy

QoS Criterion	**AV**	**SS**	**RL**	**CL**	**BP**	**Doc**
MAPPMatching	1	0	0	0	0	1
Compound2	0	1	0	1	1	0

After doing all the comparisons our final ranking order is shown in Table 3.20:

Table 3.20: Final Ranking order based on MCD strategy

Ranking Order	Web Services
1st	CasUsers
2nd	GBNIRHolidayDates
3rd	USDAData
4th	Compound2
5th	MAPPMatching

3.5.3.3. Using Weighted MCD Strategy:

Weighted MCD (WMCD) strategy is the variation of MCD strategy, where we also consider the QoS weights for the ranking of web services. For WMCD first of all we normalize

the web service values as we did in WADD strategy. Then multiply the normalized values of QoS with the weight vector. We did this for every QoS value.

Then later on MCD strategy is applied on the weighted values of QoS. We followed the same steps as we did in MCD strategy. Same pair-wise comparison is done on the weighted values.

3.5.3.4. Using Lexicographic Strategy:

In Lexicographic strategy services are ranked based on the weight of the criteria. The services which have highest weight will be ranked first followed by the services which have the lower weight. We apply Lexicographic strategy on Table 3.2 with the QoS weights {Availability: 8, Successability: 8, Reliability: 8, Compliance: 6, BestPractice: 5, Documentation: 5}. For the implementation on Table 3.2 we used the following query.

```
SELECT [ServiceName],
       [Availability],
       [Successability],
       [Reliability],
       [Compliance],
       [BestPractices],
       [Documentation]
  FROM [webservices].[dbo].[QWSDataset]
  where ServiceName = 'MAPPMatching' or ServiceName ='Compound2'
  or ServiceName = 'USDAData' or ServiceName = 'GBNIRHolidayDates'
  or ServiceName = 'CasUsers'
  order by Availability desc, Successability desc, Reliability desc,
  Compliance desc, BestPractices desc, Documentation desc
```

After applying the QoS weights on Table 3.2, the ranking order of the services in Table 3.2 will be changed as shown in Table 3.21.

Table 3.21: Ranking order with original values after the implementation of LEX strategy

QoS Criterion	**AV**	**SS**	**RL**	**CL**	**BP**	**Doc**
GBNIRHolidayDates	98	100	67	78	82	89
Compound2	95	98	67	100	83	3
USDAData	89	96	73	78	80	96
MAPPMatching	89	90	73	78	80	32
CasUsers	87	95	73	89	62	93

So the final ranking order is shown in Table 3.22, which is based on LEX strategy and this ranking order is purely based on the preferences of users. If the user changes his/her preferences the ranking order will be different.

Table 3.22: Final Ranking order based on LEX strategy

Ranking Order	Web Services
1st	GBNIRHolidayDates
2nd	Compound2
3rd	USDAData
4th	MAPPMatching
5th	CasUsers

3.6. Typical User Patterns of Following Multiple Strategies

A user may follow one decision strategy all the time when selecting services based on QoS criterion. However, it is more likely that a user may follow a few decision strategies and choose among them according to the context of search or the tasks service is used for. The user preferred strategy may also change over time. Below we give a few scenarios that could describe the typical patterns when users follow multiple decision strategies.

- Pattern 1: users follow one strategy all the time. In this case, users may only know one strategy, or are only comfortable with one strategy, and thus always use it.
- Pattern 2: users follow a few strategies with different probabilities. In this case, users are aware of a few decision strategies. The probability of following each strategy may depend on the context, tasks, the feasibility of the strategy, user's familiarity with the strategy, user's preference on the strategy, etc.

- Pattern 3: users follow a few strategies randomly. In this case, users are aware of a few decision strategies and use them constantly, however, without any obvious patterns or favorites.
- Pattern 4: users follow a few strategies among which some are dominating, e.g., their probabilities are much higher than the others. In this case, users have preferences on some strategies, so that they use them often, but they do not rule out other strategies and they still use them when necessary.

Among these four patterns, Pattern 4 can be considered as a special case of Pattern 2. There could be more patterns, but in this work we mainly consider these four.

3.7 System Architecture

In order to understand which decision strategies users follow when selecting services, we could either ask them to specify explicitly or we can learn implicitly from history logs. In the former case, if they have knowledge on decision strategies, they could choose directly from a list of provided options, otherwise, if they want to spend time to fill in questionnaires, the system can identify the strategy by checking their answers. Our system design facilitates both options. Figure 3.1 shows our system architecture model.

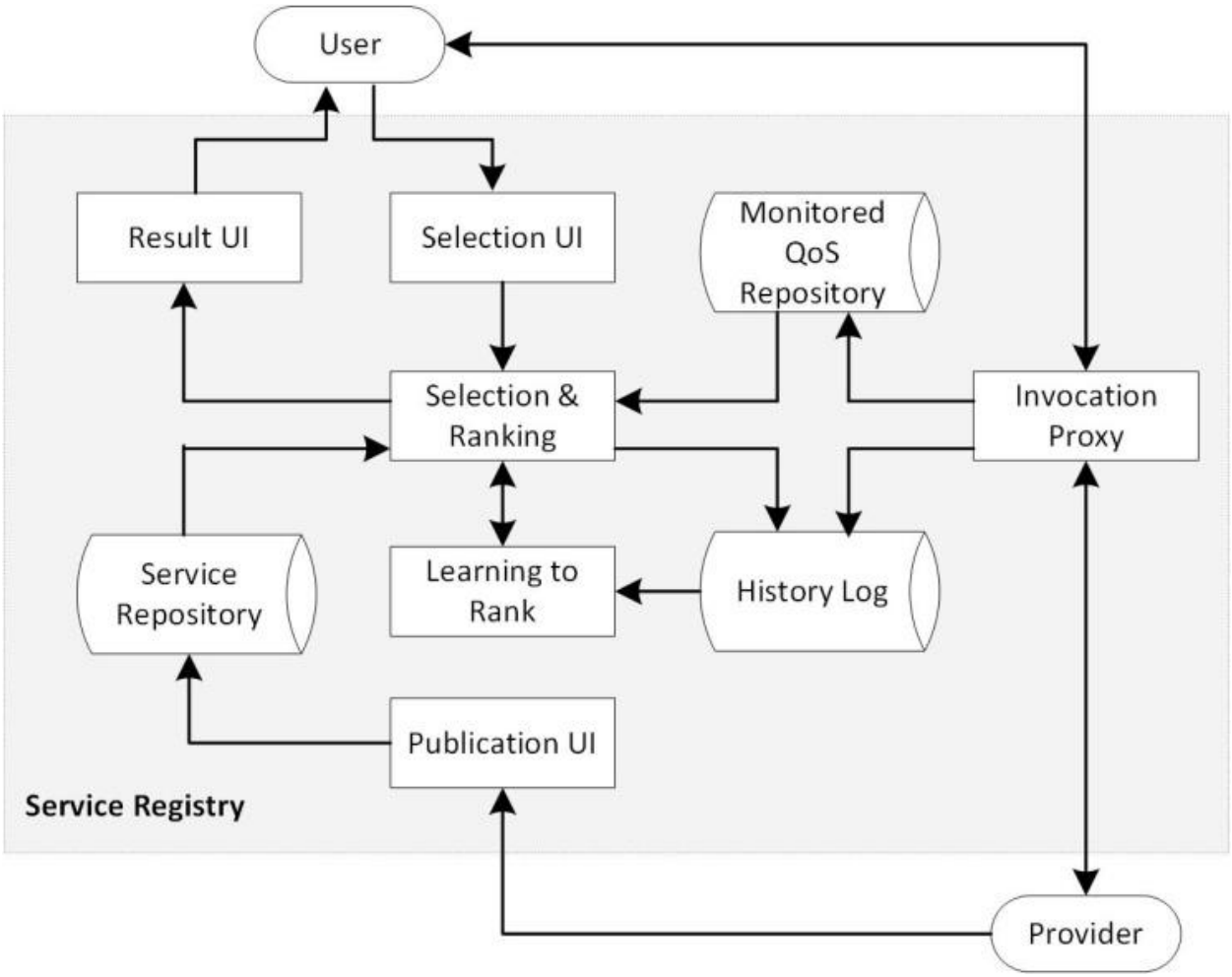

Figure 3.1. Architecture of our personalized service selection system

The selection system is part of a service registry, in which service providers can publish their services into a Service Repository. When a user is searching for a service, he enters his functional as well as QoS requirements through the Selection UI, and he also has the option to define which decision strategy to follow when selecting services. All the user input is then passed on to the Selection and Ranking Component. This component searches for the functionally matching services in the Service Repository first, and then rank these services based on the QoS requirements. If the decision strategy is explicitly defined by the user, one of the 12 decision-strategy-based ranking algorithms is used to rank the services. Otherwise, the personalized ranking algorithm learned through the Learning to Rank Component is used. The

ranked list of result is returned to the user through the Result UI. The user then selects a service for the later invocation.

In order for the system to learn user's service selection pattern or decision strategy pattern, the service invocation request is sent to the Invocation Proxy first, and then forwarded to the service provider. The delivered service also goes through the Invocation Proxy first and then to the user, in this way, the actual QoS data can be monitored and saved into the Monitored QoS Repository. The History Log keeps the records of all the users' search requests as well as invocation requests. Every record has the following information: user ID, query, matching services in the result list, and invoked service. With the history data, the Learning to Rank Component can learn the personalized service ranking algorithm for individual users, which could identify user's pattern on following decision strategies.

3.8 AdaRank to Learn the Personalized Ranking Algorithm

Learning to Rank [6] is a machine learning approach which has been successfully applied to ranking problems in many areas such as information retrieval, collaborative filtering, sentiment analysis, etc. In a ranking problem, there could be multiple ranking signals or features, and different signals produce different rankings. Many experiments (e.g., the famous NetFlix Grand Challenge [18]) have proved that the ensemble of multiple rankings normally gives the best result. By using the training data, the learning to rank algorithm could learn a function or model representing an optimal way of combining multiple rankings. The learned model can then be used to rank new objects.

In our personalized service ranking problem, there are multiple ranking algorithms based on different decision strategies, and since users normally do not specify what strategy they follow for each selection process, we would like to learn a ranking model which could best

mimic the way individual users switch between strategies according to the context and the tasks. The history log saved in the service registry can provide the training data for the learning algorithm. Through learning, we can identify the strategies a user follows as well as the best way of combining the corresponding ranking algorithms, and eventually provide a personalized ranking algorithm for user's service selection. This personalized ranking algorithm is adaptive because it can be constantly learned and updated when new user data is available, and it is also extensible because more decision strategies or more QoS based ranking algorithms can be fed into the learning model.

In this work, we use AdaRank [17] as our learning to rank algorithm. The reason we choose it is that it can directly optimize the metrics used in the selection system, whereas many other algorithms such as RankBoost [19] define loss functions loosely related to those metrics. The most commonly used performance metrics in the Learning to Rank algorithms include Mean Average Precision (MAP) and Normalized Discounted Cumulated Gain (NDCG) [20]. However, the dataset we could reasonably collect from the history log only has the information on the service a user actually selects for a query, without the knowledge on user's evaluation on other returned services. Therefore, we cannot use MAP or NDCG in our system. The metric we use is Mean Reciprocal Rank (MRR) [20], which measures the accuracy of ranking based on the position of the selected result in the ranked result list, the higher the position, the higher the MRR value.

Since in our system, personalized ranking is learned for each individual user, the history log is first partitioned on users. Then the training dataset for each user is represented as a collection of m records, and each record is represented as (q_i, rs_i, s_i), where q_i is the i-th query from the user, rs_i is a ranked list of returned services for query q_i, and s_i is the service the user

selects from the list rs_i. The learning to rank algorithm is going to learn a ranking function, so that the ranking scores generated for the returned services for a query can optimize the performance measure MRR. Given a query q_i, a ranked list of returned services rs_i, and the user selected service s_i, the Reciprocal Rank (RR) metric for q_i is defined as,

$$\mathrm{RR_i} = \frac{1}{\mathrm{positions}_{\ i}} \qquad \textbf{(3-4)}$$

where $positions_i$ defines the position of s_i in rs_i.

The MRR metric is defined over all queries as below,

$$\mathrm{MRR} = \frac{1}{\mathrm{m}} \sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{RR_i} \qquad \textbf{(3-5)}$$

The input to the AdaRank algorithm includes the training dataset, the performance measure function RR, and the number of rounds *T*. The algorithm runs *T* rounds and at each round, one weak ranker is generated. Eventually the final ranking model is defined as a linear combination of all the weak rankers as shown below,

$$\mathrm{f}(\vec{\mathrm{x}}) = \sum_{\mathrm{t}=1}^{\mathrm{T}} \alpha_{\mathrm{t}} \mathrm{h_t}(\vec{\mathrm{x}}) \qquad \textbf{(3-6)}$$

where $\mathrm{h_t}(\vec{\mathrm{x}})$ is a weak ranker, α_t is its weight, and *T* is the number of weak rankers.

In our implementation, the weak ranker is chosen as one of the decision-strategy-based ranking algorithms that has the optimal weighted performance among all algorithms,

$$\max_{\mathrm{k}} \sum_{\mathrm{i}=1}^{\mathrm{m}} \mathrm{P_t(i)\, RR_i} \qquad \textbf{(3-7)}$$

where *k* is the number of ranking algorithms considered in the system, which is 12 in our current implementation, P_t is the weight distribution at round *T*, and $P_t(i)$ is the weight on query q_i at round *T*. Initially equal weights are assigned. Then at each round, the weights on queries that are not ranked well by the current model are increased so that the model in the next round can rank those queries better. The pseudo-code of the AdaRank algorithm is shown in Figure 3.2.

Input: m – total number of queries;
T – number of rounds;
trainingData={(query, service rankings, selected service)};
Output: the ranking model f_T;
Algorithm:
//Declare Weight Distribution Array and initialize it for all queries
for i = 1 to m
weightDistribution[i] = 1/m;

for t = 1 to T
{
//Create a weakRanker with weightDistribution values
for k = 1 to 12
$wMRR_k = \sum_{i=1}^{m}(\text{weightDistribtion}[i] * RR_k(i))$;
Choose the strategy with max($wMRR_k$) as weakRanker_t;

//Calculate α_t
Calculate RR_t values for weakRanker_t;
$\alpha_t = \frac{1}{2}.log\frac{\sum_{i=1}^{m}(\text{weightDistribution}[i]*(1+RR_t(i)))}{\sum_{i=1}^{m}(\text{weightDistribution}[i]*(1-RR_t(i)))}$;

//Create ranking model f_t by linearly combining all weakRankers
$f_t = \alpha_1*\text{weakRanker}_1 + \ldots + \alpha_t*\text{weakRanker}_t$;

//Update the weightDistribution values
Calculate RR_{ft} values for f_t;
$\text{weightDistribution}[i] = \frac{\exp(-RR_{ft})}{\sum_{j=1}^{m}\exp(-RR_{ft})}$;
} //End of for loop

Figure 3.2. Pseudo-code of our learning to rank algorithm

3.9 Summary

In this chapter we have discussed the detail of our proposed approach which includes the service selection and ranking algorithms based on 12 decision strategies. We have implemented these decision strategies which are the combination of three rules and four existing decision strategies. For service selection and ranking we are considering both the QoS criteria and decision strategies. We also describe the patterns that the users may follow for different tasks. We

have described four different typical patterns when users follow various decision strategies. These patterns are based on the scenarios when users follow one strategy at a time, follow multiple strategies with different probabilities, follow multiple strategies randomly and follow multiple strategies with one dominating strategy.

We also describe the architecture model of our selection system in which user is required to enter or select his / her QoS requirements and decision strategies; otherwise the system can learn from the history logs. We have 12 ranking algorithms and sometimes users are not aware of which algorithm they should follow for their selection process. So we apply the learning to rank algorithm that learn the ranking model and find the best matching decision strategies and best ranking model for user's selection process.

CHAPTER 4

EXPERIMENTS

In this chapter, we first explain our experiment design and how we generate the simulated dataset, and then we compare our results with individual algorithms as well as a linearly combined ranking model, and provide some analyses.

4.1 Experiment Design

In our experiment, we assume that the functionally matching services have been identified using some existing methods and we only need to rank them based on users' QoS requirements and decision strategies. We also assume that we have already collected a certain amount of user query and selection history data. In the experiment, we mainly test the case when the explicit strategy information is not available, which means the learned personalized ranking model is used to rank services.

To verify our proposed approach, we should be able to show that 1) it is necessary to integrate the decision strategy into the service ranking model, 2) it is necessary to combine multiple strategies into the ranking model because users follow different strategies in different contexts, and 3) our personalized ranking model combining multiple strategies provides a good result. The first point has been verified in [21]. So here we mainly focus on the second and third points. We first compare the performance of our model with individual decision-strategy-based ranking algorithms. If our model has a better performance, it means that when users follow multiple strategies combination is necessary. Then we compare our model with a linearly combined ranking model. If our model achieves a better result, it means that learning to rank algorithm can produce a better way for rank combination.

Our system was implemented using C# language in Microsoft Visual Studio with .Net Framework 4. The experiment was run on a computer with AMD X2 Dual-Core Processor M320, 2.1 GHz clock speed, 4G RAM, and Windows 7 as the operating system. Microsoft SQL Server 2008 R2 was used as the database server.

4.2 Our Simulated Dataset

There is no publicly available dataset for our experiment, and it is also hard to find many users to use our system so that we could collect enough usage data in the logs. Therefore, we ran simulation to generate the dataset. In the simulated scenario, a user submits a QoS request, checks all the results returned from the system, and then selects one service based on a certain decision strategy. Since the decision making could be affected by the order of the results, the service selected by the user may not necessarily be the best service based on the strategy. We assume that the user is patient enough to review many results to find a good one so that it will be one of the top K results based on the strategy. Usually if the K value is not big, all the top K results can provide good results and thus the user is still satisfied with the selected service.

We used QWS dataset [22] as our QoS dataset, which includes 2507 services. We only choose 7 QoS properties out of the original 9, including availability, successability, throughput, documentation, compliance, best practices and reliability. For all of them, a higher value means a better service. The QoS queries were also generated based on this dataset. Each query could have requirements on multiple QoS properties. The number of properties was randomly chosen in the range of [1, 7]. The required values are based on the value ranges of that property in the QWS dataset. Some sample QoS queries are shown in the Table 4.1, in which Availability, Successability, Documentation, Compliance, Best Practices, Reliability has percentage numbers while Throughput has invokes/sec values.

Table 4.1: Generated QoS Queries

Query ID	Query
1	Throughput > 40.65, Documentation >= 38
2	Throughput > 20.54, Successability >= 80
3	Successability > 70, Throughput > 38.82, BestPractices > 68, Documentation >= 34, Compliance >= 77
4	BestPractices > 81, Throughput > 30.54, Compliance >= 77, Documentation >= 95, Successability > 98
5	Availability >= 99, Successability >= 89, Documentation > 84, Compliance >= 62, Throughput > 34.1
6	Throughput >= 35.15, Successability >= 85, Documentation >= 51, BestPractices >= 68
7	BestPractices > 62, Throughput > 23.99, Compliance >= 55
8	Documentation > 71, Availability >= 80
9	Availability > 68, Successability > 61
10	Compliance > 83, BestPractices > 85, Availability > 86, Documentation > 67, Successability > 73

There are 12 ranking algorithms considered in the thesis, and based on how they combine decision strategies and ranking rules, they are shown in the Table 4.2:

Table 4.2: User Decision Strategies and Ranking Rules

User Decision Strategies and Ranking Rules	**Abbreviations**
Lexicographic	LEX
Lexicographic + Layer Rule	LEXL
Lexicographic + Quantity Rule	LEXQ
Weighted Additive	WADD
Weighted Additive + Layer Rule	WADDL
Weighted Additive + Quantity Rule	WADDQ
Majority of Confirming Dimensions	MCD
Majority of Confirming Dimensions + Layer Rule	MCDL
Majority of Confirming Dimensions + Quantity Rule	MCDQ
Weighted Majority of Confirming Dimensions	WMCD
Weighted Majority of Confirming Dimensions + Layer Rule	WMCDL
Weighted Majority of Confirming Dimensions + Quantity Rule	WMCDQ

Users may follow multiple strategies in different ways. Table 4.3 lists all the multi-strategy following patterns of users which we considered in the experiment, together with their corresponding number of users.

Table 4.3: User Pattern of Following Multiple Strategies

Pattern Name	**Multi-Strategy Following Pattern of Users**	**Number of Users**
All1	Always use one ranking strategy	50
Uni2	Uniformly use 2 ranking strategies	50
Uni3	Uniformly use 3 ranking strategies	50
Uni4	Uniformly use 4 ranking strategies	50
Ran2	Randomly use 2 ranking strategies	50
Ran3	Randomly use 3 ranking strategies	50
Ran4	Randomly use 4 ranking strategies	50
Dom	Some ranking strategies dominate	50
Two91	Follow 2 strategies with probability 90%, 10%	10
Two82	Follow 2 strategies with probability 80%, 20%	10
Two73	Follow 2 strategies with probability 70%, 30%	10
Two64	Follow 2 strategies with probability 60%, 40%	10

In total, there are 440 users in our dataset, each user submitted 100 queries, and each query has requirements on 1-7 QoS properties. For each query, we saved the list of matching services which satisfy all its QoS requirements, the strategy user follows, and the service selected by the user based on the strategy. We make sure the number of strategies a user follows and the number of queries for each strategy match with what are specified in the user pattern. For instance, if a user always uses one strategy, then all the queries from that user use that strategy for service selection. Or if a user uses two strategies with probability 80%, 20%, then 80% of the queries use one strategy and 20% of the queries use the other one. The strategy is always randomly picked from 12 strategies.

After the dataset was generated, to apply the learning algorithm, for each user's usage data, 60% is used for training and 40% is used for testing. The metric used for result evaluation is MRR as defined earlier.

4.3 Results Compared With Single Strategy Based Ranking Algorithms

In this set of experiments, we fix the *K* value to 5, which means the service selected by the user could be one of top 5 results based on the user followed strategy. Figure 4.1 shows the comparison between our algorithm and those single strategy based ranking algorithms for all the multi-strategy following patterns. For each user, the MRR value is averaged on all the queries submitted by the user. And then for each algorithm, the MRR value is averaged on all the users with the same strategy following pattern. The MRR value for our algorithm is calculated on the testing data.

From Figure 4.1, we can see that on average our learned ranking model combining multiple strategies performs much better than the ranking model which only considers one strategy. Compared with the best performing individual algorithms, the improvement from our algorithm is from 9.35% to 59.66% for different patterns. Also the MRR value of our algorithm is very consistent across all patterns. However, none of the individual algorithms perform consistently well for all patterns. Another observation is that for each pattern, the best performing individual algorithm varies a lot. For instance, in All1, it is MCDL, in Uni3, it is WMCDQ, in Ran3, it is LEXQ, in Dom, it is WADDL, and in Two91, it is MCDQ. It shows that if we use the traditional ranking approach, considering only one strategy, it may work for some scenarios, but not all the time. Overall speaking, the best performing algorithms are from the MCD family, either one of the WMCD or MCD algorithms. The sample training data is provided in Appendix B.

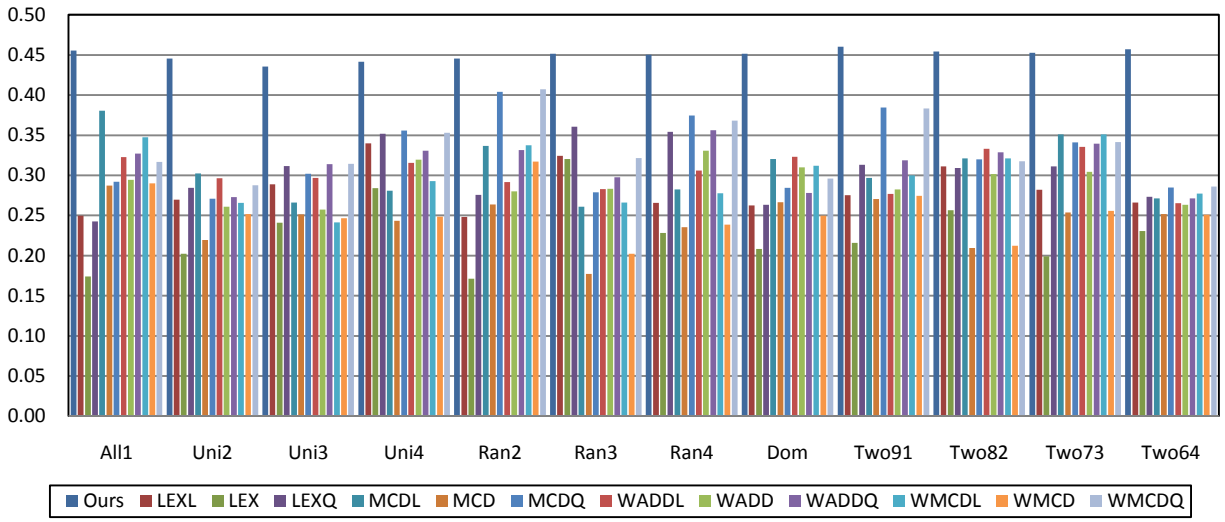

Figure 4.1. Comparison of our algorithm with individual algorithms on MRR values for all user patterns

Many QoS-based service selection algorithms use the weighted sum (WADD) as the default decision strategy. Therefore we compare the results from our algorithm with the WADD algorithm as shown in Figure 4.2. The improvement from our algorithm is obvious. So if we integrate our algorithm into any existing selection system, its accuracy can be improved.

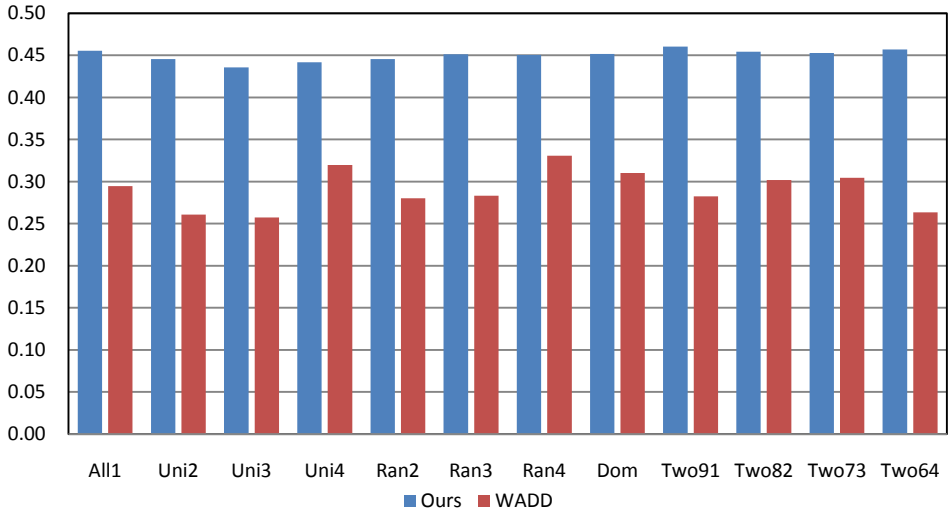

Figure 4.2 Comparison of our algorithm with WADD algorithm

Our algorithm performs well not only for average users, but also for individual users when they are following multiple strategies. Figure 4.3 shows the comparison between our algorithm and the best performing individual algorithms for 25 users who have dominating strategies (DOM). We could see that our algorithm performs much better than or close to the best performing individual algorithm. The same observation applies to all the users in DOM. However here we can only show 25 due to the space limitation.

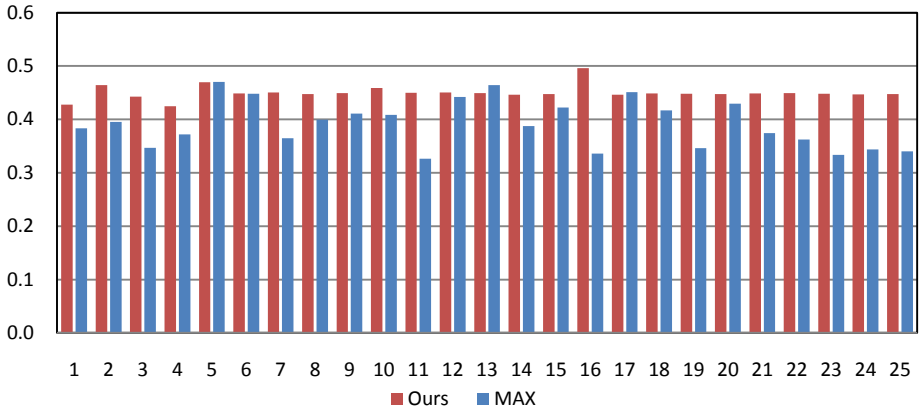

Figure 4.3. Comparison of our algorithm with best individual algorithm for 25 DOM users

Figure 4.4 shows the comparison between our algorithm and the best performing individual algorithms for 25 users who uniformly use 2 ranking strategies. We could see that our algorithm performs better than the best performing individual algorithm.

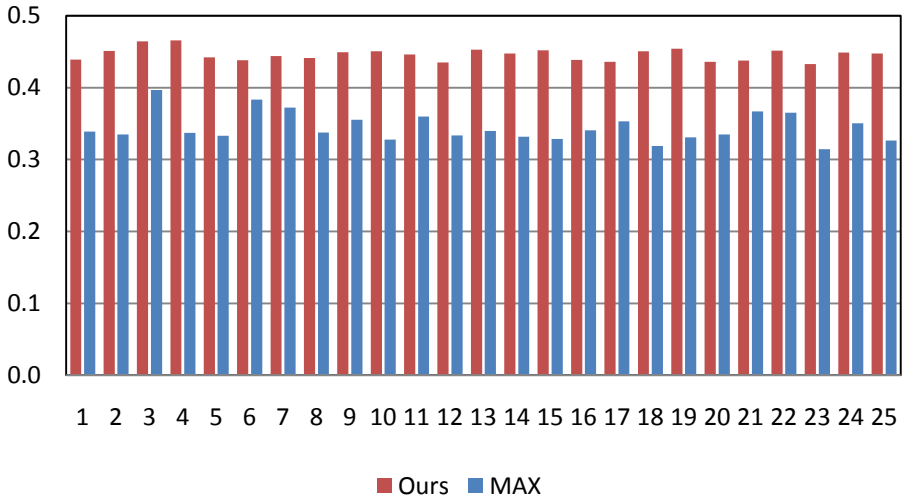

Figure 4.4. Comparison of our algorithm with best individual algorithm for 25 Uni2 users

The same conclusion can also be drawn for all the other patterns when users follow multiple strategies. However, when users only follow one strategy all the time, the result is different. Figure 4.5 shows the comparison between our algorithm and the best performing individual algorithms for 25 users who always use one ranking strategy. We could see that there is no clear winner between our algorithm and the best performing individual algorithm. The reason is that if users only follow one strategy, the ranking algorithm based on that strategy definitely performs well.

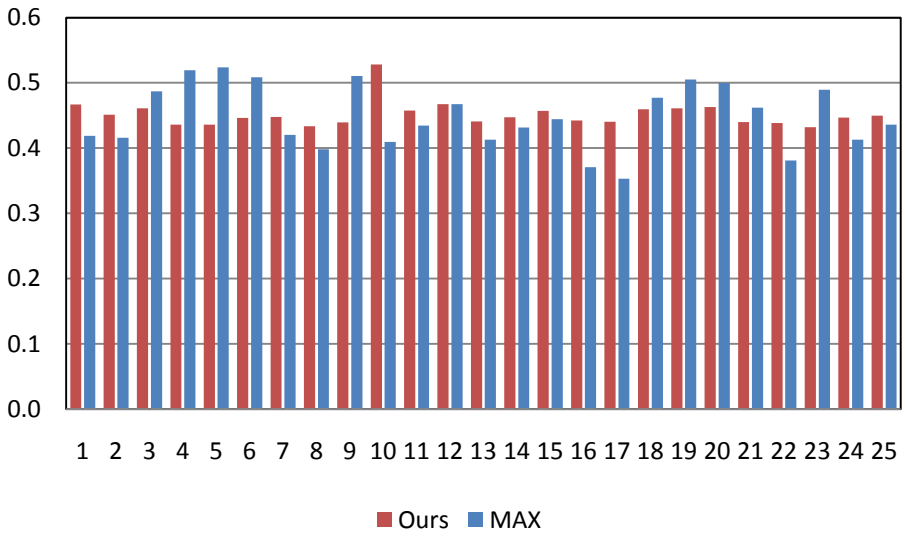

Figure 4.5. Comparison of our algorithm with best individual algorithm for 25 All1 users

4.4 Results Compared With Linear Combination Model

As we can see from the earlier results, when users follow multiple decision strategies, ranking algorithms based on a single strategy did not perform well, and therefore, it is necessary to have a ranking model considering all strategies.

Here we compare the performance of our algorithm with the ranking model which linearly combines all individual ranking algorithms. For the simplicity reason, we only consider the case when all the weights are set equal. Figure 4.6 shows the comparison result. The *K* value is still 5.

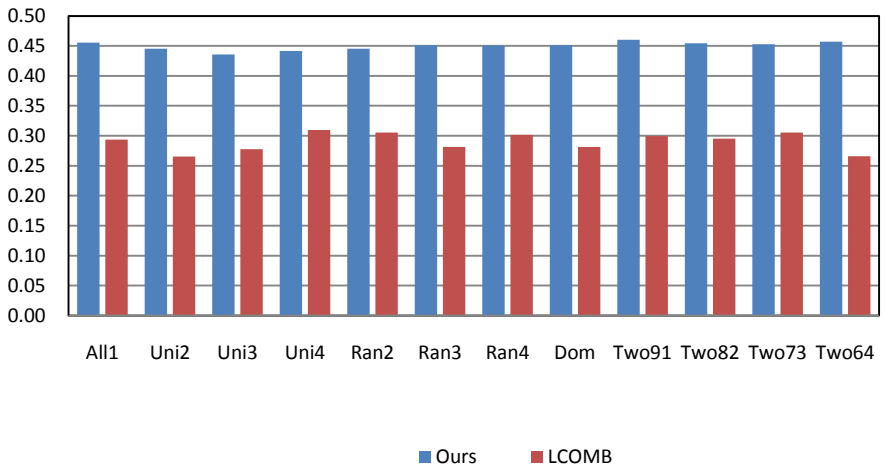

Figure 4.6. Comparison of our algorithm with linear combination algorithm

From the figure, we could see that although the linear combination algorithm combines the ranking scores from multiple individual ranking algorithms, its performance is much worse than our learned ranking model. The average improvement from our algorithm is about 50%, which shows the effectiveness of the learning step.

4.5 Impact of K Values

In the ideal case, when the *K* value is 1, which means users always choose the services which are ranked the best according to the decision strategies they follow for corresponding queries, the MRR value should be very high for a good learning algorithm. However, in reality, the selection of the service is affected by the order of the service presented to the user. Usually users would choose the first satisfying result even if it is not the best one. And also it is very likely that there are noise data in the training dataset during the learning process. Therefore, it is more realistic to use a bigger *K* value to evaluate the algorithm performance. In this set of the experiments, we want to check the impact of the *K* values. We choose 3 values: 3, 5, and 10. Since the result is consistent for all patterns, here we do not show results for different patterns separately, instead, we only show the average result on all users in Figure 4.7.

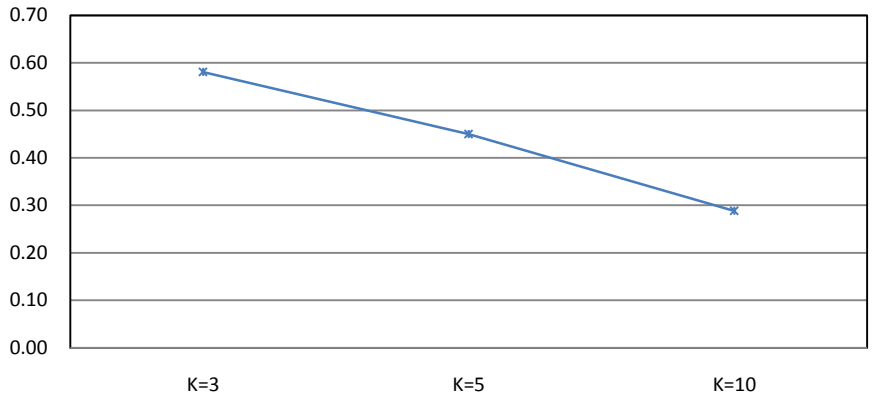

Figure 4.7. Comparison of our algorithm with different K values

We could see that when the *K* value is bigger, the MRR value becomes smaller. It is a reasonable result because when the *K* value is bigger, it means more noise is added to the training data, and thus the result is worse.

Overall speaking, our experiment results show the necessity of considering multiple decision strategies in the service selection process and the effectiveness of the learned personalized ranking algorithm.

CHAPTER 5

CONCLUSIONS AND FUTURE WORKS

5.1 Conclusions

For the functionally matching services the users often require to select and rank the services based on non-functional requirements and they can be represented as QoS criteria such as availability, reliability, throughput, response time, etc. In the selection and ranking process different users may follow different strategies which are categorized as compensatory and non-compensatory decision strategies. We improved the service selection system that reflects the user view of selecting and ranking services using his/her QoS requirements and decision strategies. We designed our customizable and flexible service selection framework for this purpose.

Our proposed framework has the following features:

- We consider that user decision strategies play a vital role in the selection and ranking of services. We implemented a personalized service ranking algorithm based on users' QoS requirements as well as their decision strategies. Different ranking algorithms were implemented for different strategies and ranking rules.
- To figure out what strategies a user follows and how they are followed, a learning-to-rank algorithm was applied on the historical data of this user's queries and subsequent selection records. A personalized ranking algorithm could then be learned through this process. We have developed a flexible approach and we can easily plug in different QoS based selection models, more user decision strategies, more QoS requirements and different learning to rank algorithms.

- As there were no public datasets available for our experiment, we have developed the application for generating the dataset where user is required to enter his/her QoS requirements, go through all the results returned from the system and then apply the particular user decision strategy on the results. We did the comparison of our results with different existing strategies and our experiment results showed the effectiveness of the proposed approach.

5.2 Future Works

QoS based web service selection is expanding and growing to include different theories and to be applied in different domains. There are a few directions we would like to work on in the future.

- Firstly, we could implement a fully functioning service selection system in a service registry, and then gradually collect the real data to test our algorithm.
- Secondly, we could integrate the decision strategy into some state-of-the-art selection models such as CP, AHP as the base selection algorithms instead of just considering some simple ranking rules.
- Finally, we would like to investigate the impact of integrating decision strategies into the service selection process in the context of service composition.

APPENDIX A – Comparison of our algorithm with other algorithms

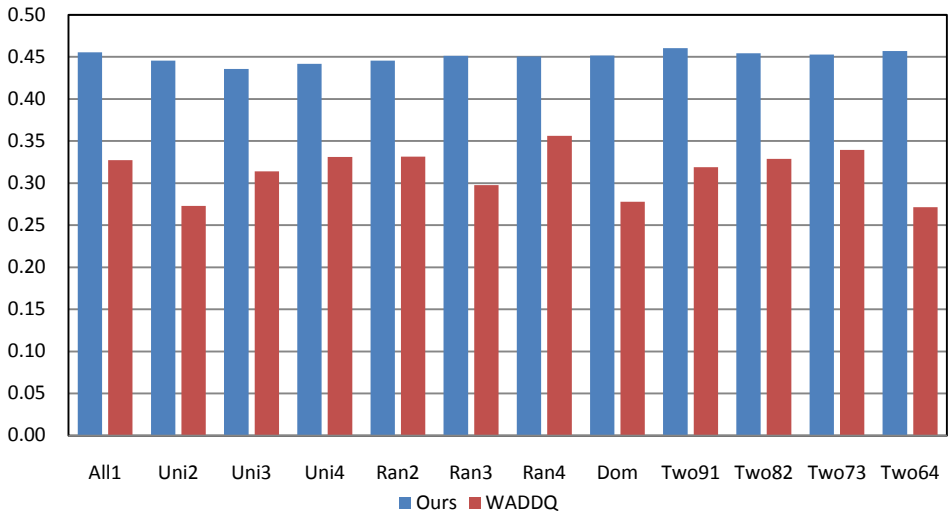

Figure A.1 Comparison of our algorithm with WADDQ algorithm

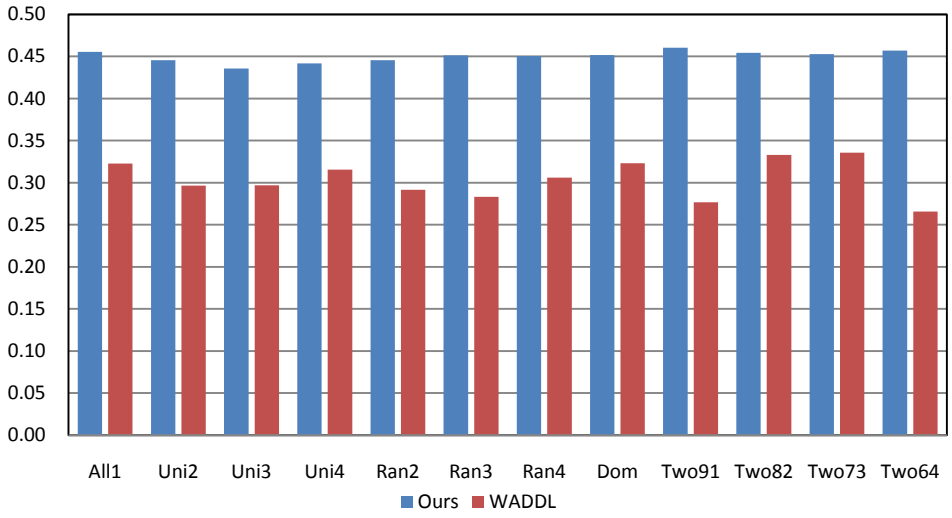

Figure A.2 Comparison of our algorithm with WADDL algorithm

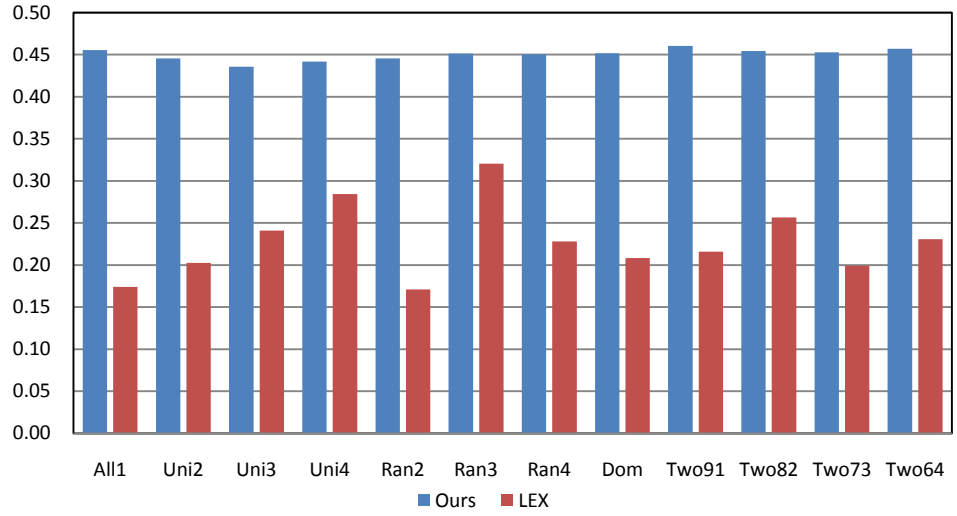

Figure A.3 Comparison of our algorithm with LEX algorithm

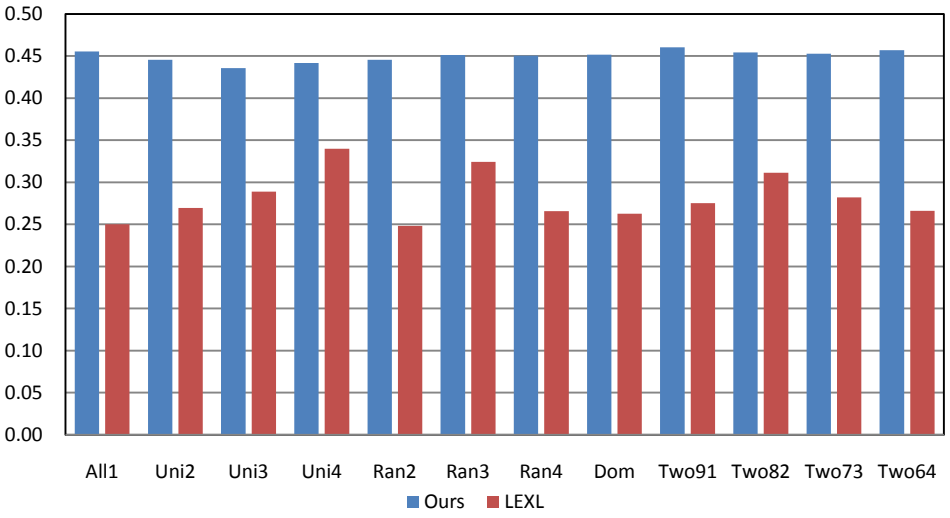

Figure A.4 Comparison of our algorithm with LEXL algorithm

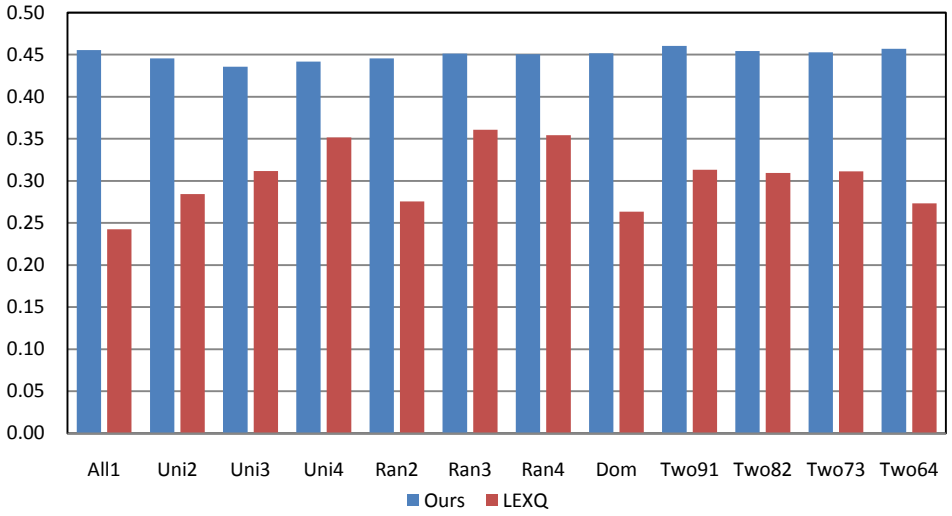

Figure A.5 Comparison of our algorithm with LEXQ algorithm

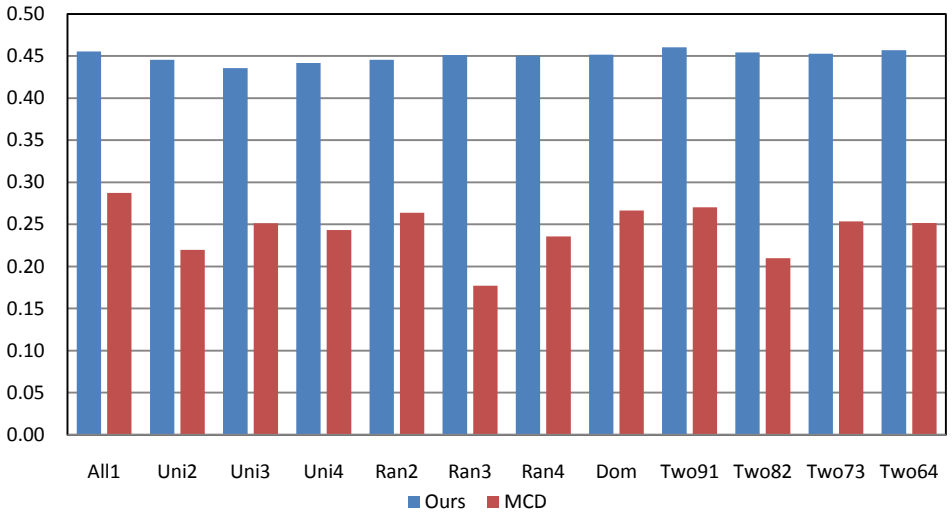

Figure A.6 Comparison of our algorithm with MCD algorithm

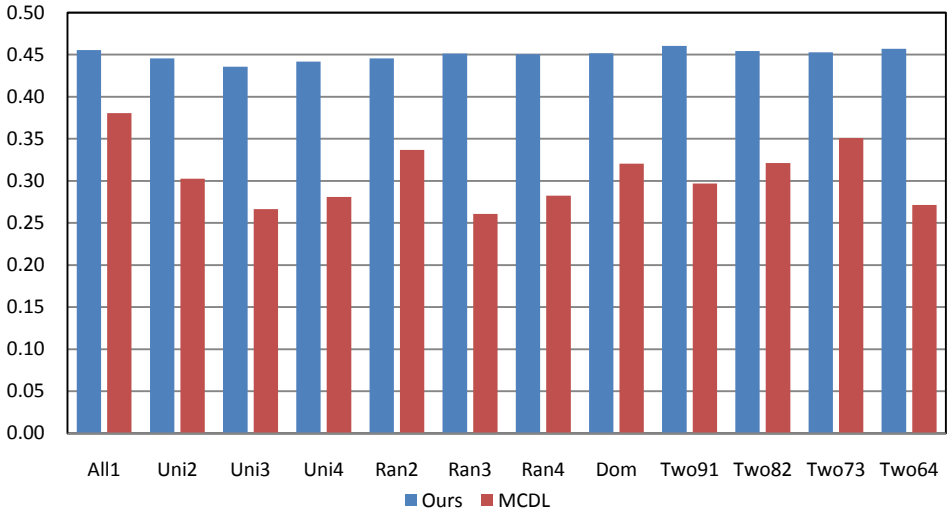

Figure A.7 Comparison of our algorithm with MCDL algorithm

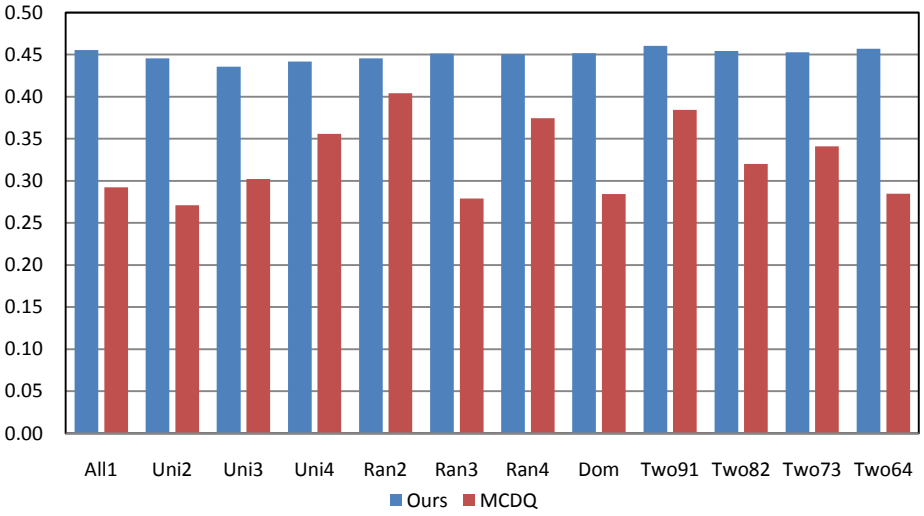

Figure A.8 Comparison of our algorithm with MCDQ algorithm

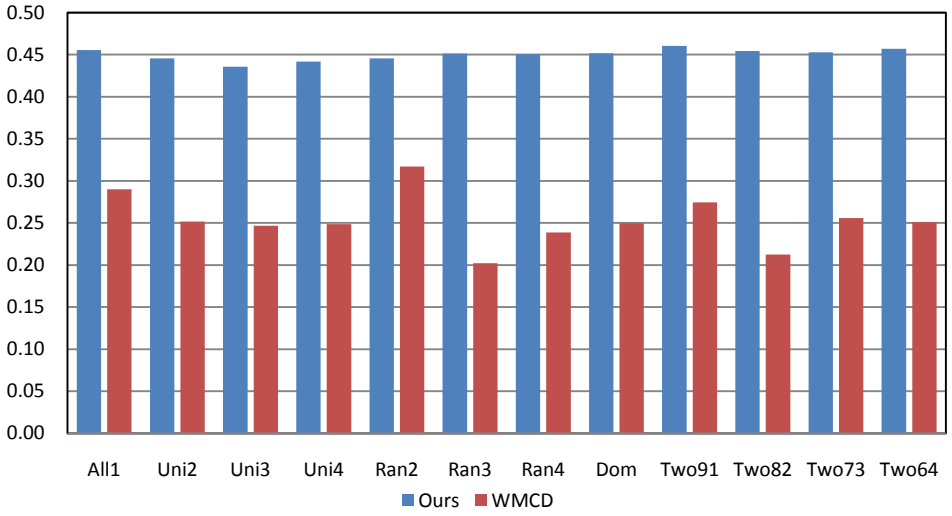

Figure A.9 Comparison of our algorithm with WMCD algorithm

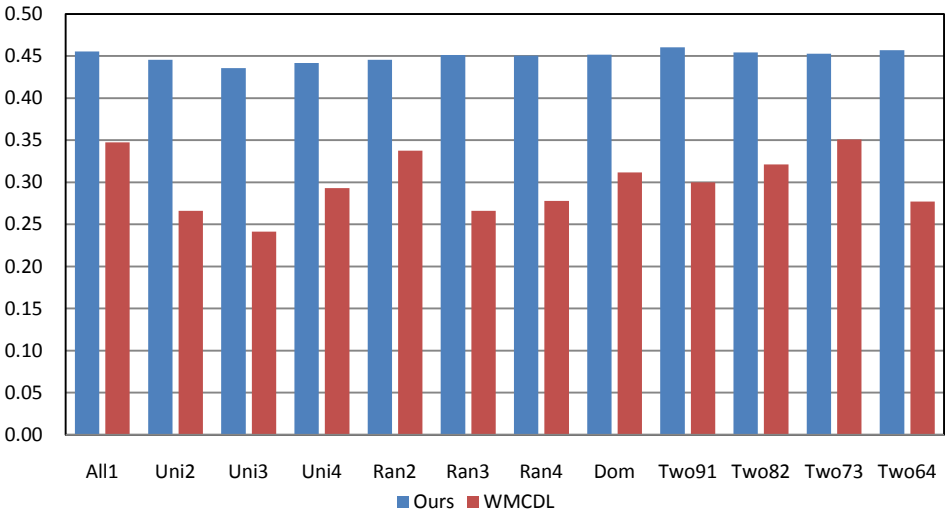

Figure A.10 Comparison of our algorithm with WMCDL algorithm

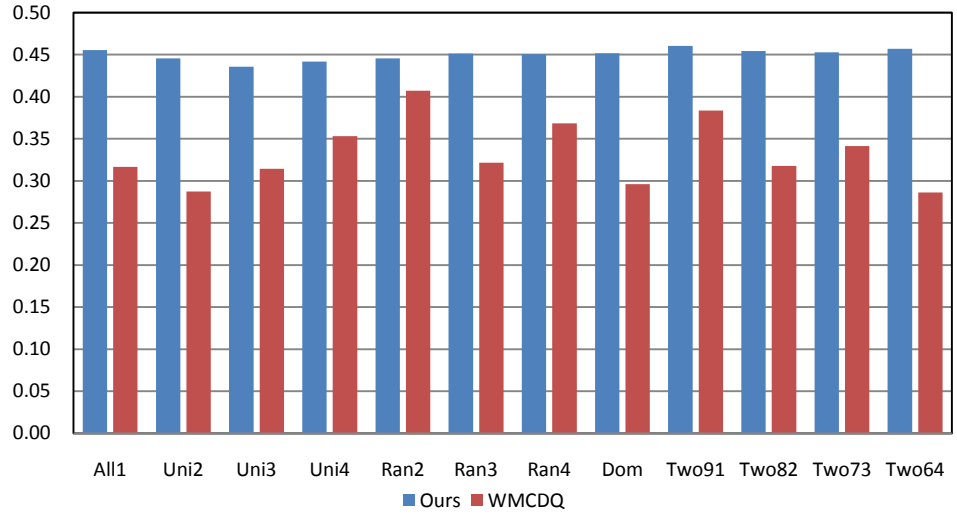

Figure A.11 Comparison of our algorithm with WMCDQ algorithm

APPENDIX B – Training Data

k=5	Ours	LEXL	LEX	LEXQ	MCDL	MCD	MCDQ	WADDL	WADD	WADDQ	WMCDL	WMCD	WMCDQ
All1	0.46	0.25	0.17	0.24	0.38	0.29	0.29	0.32	0.29	0.33	0.35	0.29	0.32
Uni2	0.45	0.27	0.20	0.28	0.30	0.22	0.27	0.30	0.26	0.27	0.27	0.25	0.29
Uni3	0.44	0.29	0.24	0.31	0.27	0.25	0.30	0.30	0.26	0.31	0.24	0.25	0.31
Uni4	0.44	0.34	0.28	0.35	0.28	0.24	0.36	0.32	0.32	0.33	0.29	0.25	0.35
Ran2	0.45	0.25	0.17	0.28	0.34	0.26	0.40	0.29	0.28	0.33	0.34	0.32	0.41
Ran3	0.45	0.32	0.32	0.36	0.26	0.18	0.28	0.28	0.28	0.30	0.27	0.20	0.32
Ran4	0.45	0.27	0.23	0.35	0.28	0.24	0.37	0.31	0.33	0.36	0.28	0.24	0.37
Dom	0.45	0.26	0.21	0.26	0.32	0.27	0.28	0.32	0.31	0.28	0.31	0.25	0.30
Two91	0.46	0.28	0.22	0.31	0.30	0.27	0.38	0.28	0.28	0.32	0.30	0.27	0.38
Two82	0.45	0.31	0.26	0.31	0.32	0.21	0.32	0.33	0.30	0.33	0.32	0.21	0.32
Two73	0.45	0.28	0.20	0.31	0.35	0.25	0.34	0.34	0.30	0.34	0.35	0.26	0.34
Two64	0.46	0.27	0.23	0.27	0.27	0.25	0.28	0.27	0.26	0.27	0.28	0.25	0.29

Table B.1 Training data of Figure 4.1

REFERENCES

[1] V.X. Tran, H. Tsuji, and R. Masuda, "A New QoS Ontology and its QoS-based Ranking Algorithm for Web Services", *Simulation Modeling Practice and Theory*, 17(8), 1378-1398, 2009.

[2] K. Kritikos, and D. Plexousakis, "Mixed-Integer Programming for QoS-based Web Service Matchmaking", *IEEE Transactions on Services Computing*, 2(2), 122-139, 2009.

[3] Q. Yu, and A. Bouguettaya, "Computing Service Skyline from Uncertain QoWS", *IEEE Transactions on Services Computing*, 3(1), 16-29, 2010.

[4] R.P. Abelson, and A. Levi, "Decision Making and Decision Theory", in G. Lindzey and E. Aronson (eds.) *The Handbook of Social Psychology*, Vol. 1. Random House, New York, 1985.

[5] J.W. Payne, J.R. Bettman, and E.J. Johnson, "Adaptive Strategy Selection in Decision Making", *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14(3), 534-552, 1988.

[6] T.Y. Liu, "Learning to Rank for Information Retrieval", *Journal of Foundations and Trends in Information Retrieval*, 3(3), 225-331, March 2009.

[7] C.W. Hang, and M.P. Singh, "From Quality to Utility: Adaptive Service Selection Framework", in *Proceedings of the International Conference on Service Oriented Computing*, pp. 456-470, 2010.

[8] K. Benouaret, D. Benslimane, and A. Hadjali, "WS-Sky: An Efficient and Flexible Framework for QoS-Aware Web Service Selection", in *Proceedings of the 9th International Conference on Service Computing*, pp. 146-153, 2012.

[9] W. Rong, K. Liu, and L. Liang, "Personalized Web Service Ranking via User Group combining Association Rule", in *Proceedings of the 7th International Conference on Web Services*, pp. 445-452, 2009.

[10] Z. Zheng, Y. Zhang, and M.R. Lyu, "CloudRank: A QoS-Driven Component Ranking Framework for Cloud Computing", in *Proceedings of the 29th IEEE International Symposium on Reliable Distributed Systems*, pp. 184-193, 2010.

[11] M.O. Shafiq, R. Alhajj, and J. Rokne, "On the Social Aspects of Personalized Ranking for Web Services", in *Proceedings of the IEEE International Conference on High Performance Computing and Communications*, pp. 86-93, 2011.

[12] Q. Zhang, C. Ding, and C.H. Chi, "Collaborative Filtering Based Service Ranking Using Invocation Histories", in *Proceedings of the International Conference on Web Services*, pp.195-202, 2011.

[13] G. Kang, J. Liu, M. Tang, X. Liu, B. Cao, and Y. Xu, "AWSR: Active Web Service Recommendation Based on Usage History", in *Proceedings of the 19th International Conference on Web Services*, pp. 186-193, 2012.

[14] R. Riedl, E. Brandstätter, and F. Roithmayr, "Identifying Decision Strategies: A Process- and Outcome-based Classification Method", *Behavior Research Method*, 40(3), 795-807, 2008.

[15] J.W. Payne, J.R. Bettman, E. Coupey, and E.J. Johnson, "A Constructive Process View of Decision Making: Multiple Strategies in Judgment and Choice", *Acta Psychologica*, 80(1-3), 107-141, August 1992.

[16] I. Jeffreys, “The Use of Compensatory and Non-compensatory Multi-Criteria Analysis for Small-scale Forestry”, *Small-scale Forest Economics, Management and Policy*, 3(1), 99-117, 2004.

[17] J. Xu, and H. Li, “AdaRank: A Boosting Algorithm for Information Retrieval”, in *Proceeding of the 30th annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 391-398, 2007.

[18] Y. Koren, “The Bellkor Solution to the Netflix Grand Prize”, 2009, *http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf* , last accessed at January 10, 2014.

[19] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer, “An Efficient Boosting Algorithm for Combining Preferences”, *Journal of Machine Learning Research*, 4, 933-969, 2003.

[20] C.D. Manning, P. Raghavan, and H. Schütze, An Introduction to Information Retrieval, Cambridge University Press, Cambridge, England, 2009.

[21] R. Karim, C. Ding, and C.H. Chi, “Multiple Non-Functional Criteria Service Selection based on User Preferences and Decision Strategies”, in *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, pp. 1-8, 2011.

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

[23] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender, "Learning to Rank using Gradient Descent", in *Proceedings of the 22nd international conference on Machine learning*, pp. 89 - 96, 2005.

[24] Y. Freund, R. Iyer, R.E. Schapire and Y. Singer, “An Efficient Boosting Algorithm for Combining Preferences”, *Journal of Machine Learning Research*, 4, 933-969, 2003.

[25] Z. Cao, T. Qin, T.Y. Lin, M.F. Tsai and H. Li, "Learning to Rank: from Pairwise Approach to Listwise Learning to Rank using Gradient Descent Approach", in *Proceedings of the 24th international conference on Machine learning*, pp. 129-136, 2007.

[26] D. Metzler and W.B. Croft, "Linear feature-based models for information retrieval", *Journal of Information Retrieval*, 10(3), 257-274, June 2007.

[27] L. Breiman, “Random Forests”, *Journal of Machine Learning*, 45(1), 5-32, 2001

[28] S. Jiang, and F. A. Aagesen, “An Approach to Integrated Semantic Service Discovery,” in *Proceedings of the First IFIP TC6 international conference on Automatic Networking*, pp. 159-171, 2006

[29] D. Booth, “Web Services Architecture”, *W3C Working Group Note* 11 February 2004, http://www.w3.org/TR/wsa-reqs/, last retrieved at July 2014.

[30] K. Kritikos, D. Plexousakis, “Requirements for QoS-based Web Service Description and Discovery”, *IEEE Transactions on Services Computing*, Vol. 2, pp. 320-337, 2009.

[31] S.W. Choi, J.S. Her, and S.D. Kim, “Modeling QoS Attributes and Metrics for Evaluating Services in SOA Considering Consumers’ Perspective as the First Class Requirement”, in *Proceedings of the IEEE Asia-Pacific Services Computing Conference*, pp. 398-405, 2007.

[32] L.T. Saaty, “Analytic Hierarchy and Analytic Network Processes for the Measurement of Intangible Criteria and for Decision-Making”, in *Multiple Criteria Decision Analysis*—

State of the Art Annotated Surveys, Vol. 78, Figueira J., Greco S., Ehrgott M., Ed. Stanford University: Springer, pp.346-406, 2005.

[33] M. Papazoglou, Web Services: Principles and Technology (1st ed.), Harlow: Pearson Education Limited, 2008.

[34] T. Erl, Service-Oriented Architecture, Concepts, Technology and Design, Prentice Hall, Indiana, 2006

[35] S. Ran.: A Model for Web Services Discovery with QoS, *SIGecom Exch.*, 4(1), 1-10 (2003)

[36] L. Clement, A. Hately, C. von Riegen, T. Rogers: UDDI Version 3.0.2, OASIS (October 19, 2004), http://uddi.org/pubs/uddi_v3.htm, last retrieved at July 2014.

[37] Inter Tech Solutions, "Service-Oriented Architecture", http://www.intertechinc.com/soa_it.html, last retrieved at July 2014.

[38] T.L. Saaty, *Theory and Applications of the Analytic Network Process: Decision Making with Benefits, Opportunities, Costs and Risks*, RWS Publications, Pittsburgh, PA, 2005.