Personalized Decision-Strategy based Web Service Selection using a Learning-to-Rank Algorithm

Muhammad Suleman Saleem, Chen Ding, Xumin Liu, and Chi-Hung Chi, *Member, IEEE*

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 in which only an overall value on all the criteria is evaluated, some evaluate one criterion at a time in the order of their importance levels, while others count on the number of winning criteria. 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 paper, 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 machine learning algorithm to learn a personalized ranking model for individual users based on how they select services in the past. We have implemented and tested the proposed approach, and our experiment results show the effectiveness of the approach.

Index Terms—Web service selection, quality of service (QoS), decision strategy, learning to rank

1 Introduction

As the number of web services published and hosted online is continuously increasing, how to select from a long list of functionally equivalent services becomes a big challenge. Automatic service selection, especially QoS-based service selection, has attracted a lot of research attentions in recent years because of its important role in facilitating a wider adoption of service computing and cloud computing technology. 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. Usually a service is considered optimal if it has the best overall QoS value among all candidates. One popular metric to evaluate the overall QoS is to take the weighted sum of multiple QoS values of a web service. Since a service may not be optimal on all the criteria, this single overall value represents the degree of compromise between the good and bad aspects of the service.

This optimization strategy may not always work in real life. According to the decision making theory [4], there are many different strategies people may follow when they decide what to choose from a list of alternatives based on multiple criteria. 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 percent, 3.5 stars), QoS values of Service 2 (*S2*) are ($85, 99 percent, 5 stars), and QoS values of Service 3 (*S3*) are ($50, 85 percent, 4 stars). And suppose there are three users *A*, *B*, and *C*, all of them have the same selection criteria: (price $\leq$ $85, reliability $\geq$ 85%, rating $\geq$ 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 at last 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 criterion—price, *S2* wins on two—reliability and rating, *S3* wins on one—price), and since *S2* is a clear winner, *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. However, when the same users select services for their work, non-compensatory strategy [5] such as

- *M.S. Saleem and C. Ding are with the Department of Computer Science, Ryerson University, Toronto, ON M5B 2K3, Canada. E-mail: {m5saleem, cding}@ryerson.ca.*
- *X. Liu is with the Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623–5603, USA. E-mail: xl@cs.rit.edu.*
- *C.H. Chi is with the CSIRO, Hobart, Tasmania, 7005, Australia. E-mail: chihung.chi@csiro.au.*

Manuscript received 1 Nov. 2014; accepted 25 Nov. 2014. Date of publication 9 Dec. 2014; date of current version 9 Oct. 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2014.2377724

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 a clear gap between the way current selection system works and the real scenario where people may follow a variety of decision strategies during the selection process. Our objective in this paper 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 time or in different context or depending on purposes of services. To take all these into consideration, we apply the machine learning 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.

In this work, 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.

There are three major contributions of the paper: 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 propose a 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 machine learning technique to find the best matching strategies and the best ranking model combining them; 3) the proposed approach is flexible and extensible so that we can plug-in different QoS-based selection models, decision strategies, as well as machine learning algorithms.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 gives the details of the proposed approach, including the definitions of the QoS properties we use in this work, the service selection and ranking algorithms based on different decision strategies, the weighting scheme used to assign priority to different properties, the constraint matching rules based on the MIP model, the architecture model of our selection system, and the learning process to obtain the personalized service ranking model. At the end of this section, we also give an illustrative example to show the whole process. Section 4 explains our experiment design, show the simulation process to generate the datasets, and analyze and discuss the results for the proposed approach. Finally Section 5 concludes the paper and lists the future directions.

2 Related Work

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], the semantic QoS description is transformed into MIP problems, and then a MIP engine is exploited for matchmaking. It is proved that MIP provides a more efficient solution than CP. However, since MIP can only solve the linear constraints, when there are non-linear constraints in the user requirements, CP is used. In [1], an Analytic Hierarchy Process (AHP) model is used. 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. Skyline computation is used for service selection in [3]. It finds optimized solutions by measuring their dominance relationships. 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.

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 using the collaborative filtering technique to find similar users based on their invocation histories and the association rule mining to identify service dependencies based on the past composition transactions of similar users. In [10], user similarity is measured as the similarity between the rankings of their observed QoS values on commonly invoked services, and the personalization is implemented using the past experiences from similar users. In [11], previous interactions between service providers and requestors are modeled as a social network, and then 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. In [13], both functional interests and QoS preferences of users from their usage history are considered for similarity calculation and service recommendation.

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.

There are many strategies people may follow when they make decisions. In [5], two types of strategies are discussed, compensatory and non-compensatory. In [14], more strategies are discussed and they are classified based on their characteristics, such as whether all attribute values are processed, whether they are attribute based or option based, whether they eliminate options before the final decision, etc. According to [15], decision makers may use multiple strategies for making choices and they 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 number of alternatives available and the different characteristics of the decision

problem. 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].

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.

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. And its main purpose is to automatically construct a ranking model using training data for ranking new objects or entities. Learning to rank approaches can be divided into three types, which are point-wise, pair-wise and list-wise approaches, based on the types of input objects for learning. If we use document ranking as an example, in a point-wise approach, the input object is a single document and it requires a relevance judgment for each document in the training set; in a pair-wise approach, the input object is a pair of documents and it requires a relevance judgment for each pair of documents; and in a list-wise approach, the input object is a list of documents and it requires a relevance judgment for a list of documents. Point-wise and pair-wise approaches do not consider interdependence between documents and hence position information is missing, whereas list-wise approaches consider the position information in the resulting ranked list. It has been proved [6] that the list-wise ranking algorithms give better performance than the point-wise and pair-wise algorithms.

3 Personalized Web Service Selection

In this section, first we explain several common QoS properties which we use in this work, the decision strategies we consider in the service selection process, the weighting scheme used for prioritizing the QoS propertyes, and the MIP-based constraint matching rules for checking the QoS constraints. Then we show a few scenarios where a user may follow different decision strategies or using different ranking algorithms for selecting services. After that, we describe our system architecture model, especially on how we collect the history data, because only when users' previous service selection requirements and results are available, machine learning technique can be applied to find out the best ensemble approach of combining different decision-strategy based service ranking algorithms. Lastly, we discuss how we use a learning-to-ranking model to learn our personalized service selection algorithm. We use an example to illustrate the whole process at the end.

3.1 QoS Properties of Web Services

A web service has many QoS properties which can be used for service selection [17]. Since we are using QWS dataset [18] in our experiment, here we list the definitions of the QoS properties whose values are collected in QWS dataset.

- *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 particular 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 such as the description tags in WSDL files. It is measured in (%).

In our selection system, out of these 9 QoS properties, we use 7 of them, including Availability, Throughput, Successability, Reliability, Compliance, Best Practice, and Documentation. Our selection system is flexible and we can add more QoS properties if necessary.

3.2 Service Selection based on Decision Strategies

Most of the current QoS-based service selection systems only consider users' QoS requirements and QoS preferences in the selection and ranking process. However, as we know, the strategies users follow when making their decisions should also play an important role in the selection process. In many selection systems, usually services are ranked based on scores on their compromised overall QoS values. This type of compensatory ranking process is just one of many decision strategies users may follow. If we simply use it for all users, the ranking result may not be accurate for individual users. In our selection system, we want to consider multiple decision strategies and the ranking results are based on both strategies and user requests.

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 MCD (WMCD) strategy, and Lexicographic (LEX) strategy [5]. Since decision strategies only target at finding one optimal solution, in order to generate a full ranking order on services, we modify the original processes of these decision strategies. Below, we give explanations on ranking algorithms based on these four decision strategies.

In the WADD-based ranking algorithm, 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 below,

$$score(s_i) = \sum_{k=1}^{M} w_k \cdot a_{ik}, \quad (1)$$

where k represents the kth 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 kth property. This is the ranking algorithm used by many service selection systems [1], [2], [3]. It is used by user *C* in our earlier example.

In the MCD-based ranking algorithm, a service with a majority of winning (better) QoS values is ranked higher when compared with another service. Suppose there are two services s_i and s_j, based on the MCD strategy,

If $\sum_{k=1}^{M} p_{ijk} > \sum_{k=1}^{M} p_{jik}$.

Then s_i is ranked higher than s_j.

Where p_{ijk} is set to 1 if s_i is better than s_j on the kth QoS property, otherwise is 0. We do the comparison on all pairs of services to get the complete ranking order of these services. It is used by user *B* in our earlier example.

WMCD is a variation of the MCD strategy in which a weight is associated with each decision making criterion. Based on the WMCD strategy,

If $\sum_{k=1}^{M} w_k \cdot p_{ijk} > \sum_{k=1}^{M} w_k \cdot p_{jik}$.

Then s_i is ranked higher than s_j

In the LEX-based ranking algorithm, 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. It is used by user *A* in our earlier example.

3.3 Weighting Scheme

In the four decision strategies we consider in our system, two of them (WADD and WMCD) need to use weights of the criteria in the score calculation. LEX strategy also needs weights to decide the order of comparison. Weight tells how important a criterion is to a certain user. For a more important criterion, it will play a bigger role in the decision making process.

Through the interface of our selection system, users are allowed to specify the preference levels of QoS properties. We follow a scaling scheme of relative measurement as proposed in [19] for assigning preferences to QoS properties. The range of the scale is from 1 to 9, in which 1 represents the least important property and 9 represents the most important property. Table 1 shows 9 scales along with their linguistic descriptions [19].

If no preference is assigned to a QoS property, then zero weight will be assigned to that particular QoS property, which shows that the user has no preference over this property. By only asking users to specify the preference levels instead of the weights, our system put less stress on users because usually it is hard to pick a right number (i.e., a real number between 0 and 1) to represent the importance level of a QoS property. Our system also gives flexibility to users to define their own preference orders instead of taking default system-picked values.

TABLE 1
Scales indicating Importance Levels of QoS Properties

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

The next step is to calculate a weight according to the chosen preference value. The following simple weighting formula is used,

$$w_k = \frac{q_k}{\sum_{h=1}^{M'} q_h}, \quad (2)$$

where w_k is the weight of the kth QoS criterion, q_k represents the user defined preference of the kth criterion and M' represents the total number of criteria the user has selected.

For both WADD and WMCD, weights need to be calculated using Equation (2), while for LEX, only preference value is required.

3.4 Constraint Matching Rules based on MIP

In a user's QoS requirement, there could be constraints on some criteria, such as reliability $> 90\%$, 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. Since most of the decision strategies only target at finding optimal solutions without handling the constraints, we need to define the constraint matching rules which could be used to rank services based on how well they satisfy the constraints.

Both CP and MIP have been successfully used for QoS constraint checking and service selection in previous research work [2], [20], [21], and according to [2], MIP approach is more efficient compared to CP. Therefore, we choose to use MIP as the basis for our constraint matching rules. Although MIP does not support non-linear constraints and has other disadvantages, its high efficiency and the ability to handle both integer and real value variables make it a good option in our current setting.

Given QoS criteria (or constraints) defined by a user, a service is said to be conforming to the constraints if QoS values guaranteed by the service meets the values required by the user [20]. For instance, if the requirement is (reliability $> 90\%$, 1s $<$ response time $<$ 2s), and the QoS guarantee from the service is (reliability $> 95\%$, 1s $<$ response time $<$ 1.5s), since every single QoS value in the guaranteed range satisfies the requirement, we say that this service is conforming to the constraints. To deal with the over-constrained requirements as well as the scenario where none of the services can match all the constraints, and to increase the probability of having a non-empty solution set, we also consider partial matches which conform to some criteria but

not all. Similar to [2], there are four categories of services, among which the first two are the conforming results and the other two are the non-conforming results:

1) S*uper*—if a service satisfies all the QoS constraints and offers better-than-the-required QoS values on at least one property, then it is called a *super* match. For instance, as in our example in Section 1, the selection criteria are: price $\leq$ \$85, reliability $\geq$ 85%, and rating $\geq$ 3. If a service has QoS values as price $\leq$ \$85, reliability $\geq$ 95% and rating $\geq$ 3, it is considered as a *super* match since it satisfies all the constraints and has better value on reliability.
2) *Exact*—if a service satisfies all the QoS constraints, it is called an *exact* match. In the above example, if a service has QoS values as price $\leq$ \$85, reliability $\geq$ 85%, and rating $\geq$ 3, it is considered as an *exact* match.
3) *Partial*—if a service satisfies some of the QoS constraints but not all of them, it is called a *partial* match. In the above example, if a service has QoS values as price $\leq$ \$85, reliability $\geq$ 75%, and rating $\geq$ 3, it is considered as a *partial* match since it violates the constraint on reliability.
4) *Fail*—if a service does not satisfy any of the constraints, it is a *fail* result. In the above example, if a service has QoS values as price $\leq$ \$95, reliability $\geq$ 75%, and rating $\geq$ 2, it is considered as a *fail* result since it violates all three constraints.

Although in the above example, the QoS constraints are very simple, MIP model has the capability of processing more complicated constraints such as the ones involving both upper and lower bounds. If the system needs to support the non-linear constraints, similar to [2], we may use CP model to process them. Using the MIP (or CP) model for the constraint checking gives us a formal basis to process the constraints, and also a MIP Problem Solver can be used to automatically perform the conformance checking and get a solution.

Currently, we do not differentiate between the *super* and *exact* categories because of the way we generate the simulated dataset (e.g., the queries are randomly generated with the values falling into a valid range based on QWS dataset [18] and there are very few *exact* matches for these random queries). Therefore we have 3 categories of results: category 1—services satisfying all of the constraints (*super* and *exact*), category 2—services satisfying some of the constraints (*partial*), and category 3—services satisfying none of the constraints (*fail*). The ranking order of these three categories is category 1 $>$ category 2 $>$ category 3. This is our first constraint matching rule, which is called Layer Rule. In the MIP model used in [2], an objective function is defined to rank services. However, here we mainly use the constraint checking feature of the MIP model, not the ranking function.

The second rule is Quantity Rule. Based on this 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. Compared to Layer Rule, this rule has a stronger enforcement on the constraint matching. The reason we define this rule is that we would like to check whether constraints should play a bigger role in the service ranking process.

In our system, the actual service ranking algorithm is based on 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 applied, 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. In the rest of this paper, we usually use the term "decision strategy" to refer to the combination of a decision strategy and a constraint matching rule, unless we emphasize on the decision strategy alone.

3.5 Typical User Patterns of Following Multiple Strategies

A user may follow one decision strategy all the time when selecting services based on QoS criteria. 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 paper we mainly consider these four.

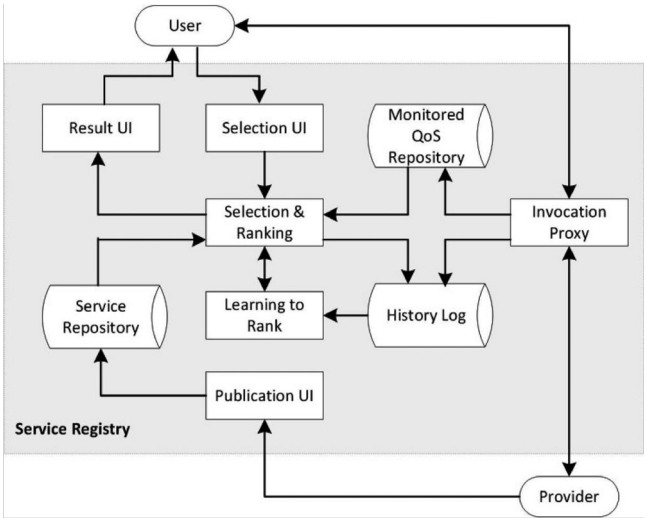


Fig. 1. Architecture of our personalized service selection system.

3.6 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. Fig. 1 shows our system architecture model.

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 learned algorithm can reflect user's preference. If a strategy contributes more to the final ranking score, it means it is preferred by the user. The ranked list of results 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 making 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.

3.7 AdaRank to Learn the Personalized Ranking Algorithm

In a ranking problem, there could be multiple ranking signals or features, and different signals produce different rankings. Many experiments (e.g., the NetFlix Grand Challenge [22]) 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 and more QoS based ranking algorithms can be fed into the learning model.

In this paper, we use AdaRank [23] as our learning to rank algorithm. One reason we choose it is that AdaRank is a list-wise learning to rank algorithm, and as we mentioned earlier, list-wise algorithm usually provides a better result than the other two types of algorithms. Another reason we choose it is that it can directly optimize the metrics used in the selection system, whereas many other algorithms such as RankBoost [24] 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) [25]. Since 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, we cannot use MAP or NDCG in our system. The metric we use is Mean Reciprocal Rank (MRR) [25], 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 ith 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. A feature vector $\overrightarrow{x_{ij}} = \Psi(q_i, rs_{ij}) \in \chi$ is defined for each query-service pair,

where rs_{ij} represents the jth service in the ranked list rs_i, and the score on each feature in our case is the ranking score calculated using one of the decision strategy based service ranking algorithms. The learning algorithm is going to learn a ranking function: $f : \chi \mapsto \Re$, 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 permutation π_i on rs_i, and the user selected service s_i, the Reciprocal Rank (RR) metric for q_i is defined as,

$$RR_i = \frac{1}{\pi_i(s_i)}, \tag{3}$$

where $\pi_i(s_i)$ defines the position of s_i in rs_i.

The MRR metric is defined over all queries as below,

$$MRR = \frac{1}{m}\sum_{i=1}^{m} RR_i. \tag{4}$$

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,

$$f(\vec{x}) = \sum_{t=1}^{T} \alpha_t h_t(\vec{x}). \tag{5}$$

where $h_t(\vec{x})$ 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,

$$max_k \sum_{i=1}^{m} P_t(i) RR_i, \tag{6}$$

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 Fig. 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
$wMRR_k = \sum_{i=1}^{m}(weightDistribtion[i] * RR_k(i))$;
Choose the strategy with max($wMRR_k$) as $weakRanker_t$;

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

//Create ranking model f_t by linearly combining all weakRankers
$f_t = \alpha_1$*$weakRanker_1$ + … + α_t*$weakRanker_t$;

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

Fig. 2. Pseudo-code of our learning to rank algorithm.

3.8 An Illustrative Example

Suppose a user is looking for a flight checking and reservation web service. He submits this functional request to our system. Since there are a number of functionally matching services, he also submits the QoS requirements to further rank the candidate services. In this example, we consider six QoS propreties out of the nine defined earlier. All of the properties have a positive tendency, which means a bigger value indicates a better option. The unit for all of them is %. Table 2 shows the user requirements on these six QoS properties as well as the QoS values from the five candidate web services.

In the table, AV stands for Availability, SS stands for Successability, RL stands for Reliability, CL stands for Compliance, BP stands for Best Practices, and Doc stands for Documentation. FS1, FS2, FS3, FS4 and FS5 represent the matching services.

Suppose the user-defined preferences on these properties are: (AV: 8, SS: 8, RL: 8, CL: 6, BP: 5, Doc: 5). Using Equation (2), their corresponding weights are: (AV: 0.2, SS: 0.2, RL: 0.2, CL: 0.15, BP: 0.13, Doc: 0.13).

We first consider the WADD strategy. Although in this particular example, all the QoS properties have the same unit and their values are all in the [0, 1] range, it is very likely different QoS properties have different value ranges. Therefore, the first step is to normalize the QoS values. We use the idealizing approach [19] for normalization. The idealizing approach is applied on each criterion by assigning 1 to the web service with the highest value and then divides the value of each remaining service by this highest value. This process is done for all the QoS criteria. The next step is to apply the weighted sum method to get the ranking score. This is done by multiplying the weight vector with the normalized values of QoS properties, and then at the end adding up all the weighted QoS values of each service to get the final ranking score. The calculation steps for 5 services are shown below.

$$\begin{aligned}\text{WADDscore(FS1)} &= 0.91*0.2 + 0.9*0.2 + 1*0.2 \\ &\quad + 0.78*0.15 + 0.95*0.13 \\ &\quad + 0.33*0.13 = 0.84 \\ \text{WADDscore(FS2)} &= 0.87*0.2 + 0.95*0.2 + 1*0.2 \\ &\quad + 1*0.15 + 1*0.13 \\ &\quad + 0.02*0.13 = 0.843\end{aligned}$$

TABLE 2
List of Functionally Matching Services

QoS Criteria	AV	SS	RL	CL	BP	Doc
User Request	>88	>= 96	>70	>80	>= 82	>= 60
FS1	89	90	73	78	80	32
FS2	85	95	73	100	84	2
FS3	89	96	73	78	80	96
FS4	98	100	67	78	82	89
FS5	87	95	73	89	62	93

$$\begin{aligned}\text{WADDscore}(\text{FS3}) &= 0.91 * 0.2 + 0.96 * 0.2 + 1 * 0.2 \\ &\quad + 0.78 * 0.15 + 0.95 * 0.13 \\ &\quad + 1 * 0.13 = 0.94 \\ \text{WADDscore}(\text{FS4}) &= 1 * 0.2 + 1 * 0.2 + 0.92 * 0.2 \\ &\quad + 0.78 * 0.15 + 0.98 * 0.13 \\ &\quad + 0.93 * 0.13 = 0.95 \\ \text{WADDscore}(\text{FS5}) &= 0.89 * 0.2 + 0.95 * 0.2 + 1 * 0.2 \\ &\quad + 0.89 * 0.15 + 0.74 * 0.13 \\ &\quad + 0.91 * 0.13 = 0.93.\end{aligned}$$

Based on the calculated scores, the order of these five services is FS4 > FS3 > FS5 > FS2 > FS1.

MCD strategy requires a pair-wise comparison. QoS values of two services are compared and the service with a higher number of winning QoS values will be selected and then the winning service will be compared with the next service. This process continues until all the services are compared. In this example, the first pair of services is FS1 and FS2. FS1 wins on 2 QoS criteria (AV and Doc) and FS2 wins on 3 QoS criteria (SS, CL, and BP). And thus FS2 is the winning service and moves on to the second round. Then we compare FS2 with FS3, and the latter one wins. Next, FS4 wins over FS3, and is then compared with FS5. In this round of comparison, both services win on 3 criteria, and thus there is a tie. In this case, we choose the winning service as the one that has a better value on the last criterion (in this example Doc). Therefore FS5 is the winning service and it is ranked # 1 among the list of services. In the next phase we remove this winning service from the list of services and do the comparison for the rest of the four services. We continue this process until we get the final ranking order, which is FS5 > FS4 > FS3 > FS2 > FS1.

WMCD strategy is the variation of the MCD strategy, where we also consider the QoS weights for the ranking of web services. For WMCD, we first normalize the web service values as we did for WADD strategy. Then we multiply the normalized QoS values with the weight vector. We do this for every QoS criterion. Afterwards, MCD strategy is applied on the weighted QoS values. We follow the same steps as we did for MCD strategy. Here the pair-wise comparison is done on the weighted values. For this particular example, we get the same ranking order as the one for MCD.

In the LEX strategy, services are ranked based on the weight of the criteria. AV has the highest preference value 8, and thus all the services are ranked on AV first. The ranking order is FS4 > FS1, FS3 > FS5 > FS2. Since there is a tie between FS1 and FS3, these two services are ranked on the criterion with the second highest value, which is SS in this case. According to their SS values, FS3 is better than FS1. The final order is FS4 > FS3 > FS1 > FS5 > FS2.

Now let us consider the constraint matching rules. If we apply Layer Rule, all 5 services are in category 2 (*partial* matches). If we apply Quantity Rule, the order of the services is FS3, FS4 (satisfying 4 constraints) > FS2, FS5 (satisfying 3 constraints) > FS1 (satisfying 2 constraints). So if we combine MCD strategy with No Rule, the final ranking order is FS5 > FS4 > FS3 > FS2 > FS1. If we combine MCD strategy with Layer Rule, we get the same ranking order as the one for No Rule because all the services are in the same category. If we combine MCD strategy with Quantity Rule, the final ranking order is FS4 > FS3 > FS5 > FS2 > FS1.

From this example, we could see that when a user follows different decision strategies, different ranking orders are generated. If we could accumulate a certain amount of historical data on QoS queries submitted by the user along with the service selected by the user for each query, we could figure out whether this user always follows one strategy, or sometimes follows one strategy and sometimes follows another strategy, or randomly picks a result, etc., through our learning-to-rank algorithm. And then we can use this learned personalized model for future service ranking. As long as the amount of historical data is adequate, the learned model should offer a good result catered to individual users.

4 Experiments

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

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) decision strategy based ranking algorithm alone is not enough to produce an accurate result, and thus it is necessary to also do the constraint checking, 3) it is necessary to combine multiple strategies into the ranking model because users follow different strategies in different contexts, and 4) our personalized ranking model combining multiple strategies provides a good result. The first point has been verified by our earlier work [26]. So here we mainly focus on the second, third and fourth points. We first compare the performance of the ranking algorithm when the constraint matching rules are considered and when they are not. If the ones with the constraint checking

TABLE 3
Sample QoS Queries

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

TABLE 4
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

component implemented have better results, it means we need to combine decision strategy with constraint matching rules. Then we compare our model with individual decision-strategy-based ranking algorithms. If our model has a better performance, it means the combination is necessary when users follow multiple strategies. At last 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, 4 GB 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 (i.e., $\leq$ 10), all the top K results can provide good results and thus the user is still satisfied with the selected service.

We used QWS dataset [18] as our QoS dataset, which includes 2,507 services. We only chose 7 QoS properties out of the original 9, including Availability, Reliability, Successability, Throughput, Documentation, Compliance, and Best Practices. All of them have a positive tendency, and thus 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 Table 3, in which the unit of Throughput is invokes/sec, and all the others have percentage values.

There are 12 basic ranking algorithms considered, and based on how they combine decision strategies and constraint matching rules, they are represented as LEX, LEXL (LEX + Layer Rule), LEXQ (LEX + Quantity Rule), WADD, WADDL, WADDQ, MCD, MCDL, MCDQ, WMCD, WMCDL, WMCDQ respectively. As mentioned earlier in Section 3.5, a user may follow multiple strategies in different ways. Table 4 lists all the multi-strategy following patterns of users which we considered in the experiment, together with their corresponding number of users we simulated.

In the generated 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 percent, 20 percent, then 80 percent of the queries use one strategy and 20 percent 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 percent is used for training and 40 percent is used for testing. The metric used for result evaluation is MRR as defined earlier.

4.3 Results Comparing Different Constraint Matching Rules

In this set of experiments, we fix the K value to 5, which means the service selected by the user could be one of the

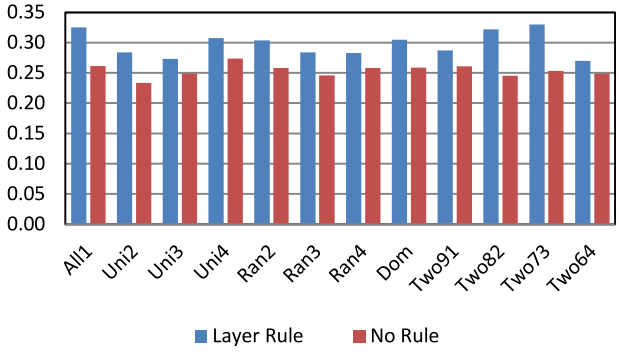


Fig. 3. Comparison between Layer Rule and No Rule.

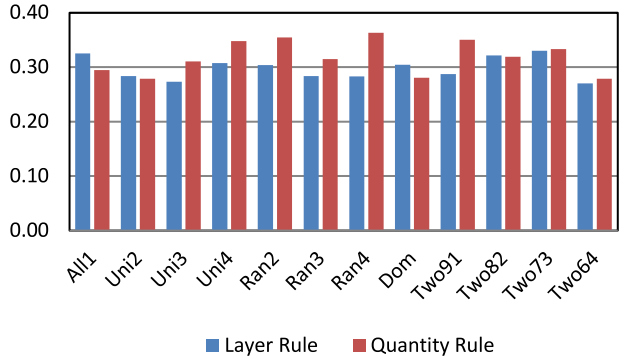


Fig. 4. Comparison between Quantity Rule and Layer Rule.

top 5 results based on the user followed strategy. Fig. 3 shows the comparison between average MRR values from the approaches using Layer Rule and the ones using No Rule for all user 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. Finally the MRR value is averaged on all algorithms using Layer Rule (LEXL, WADDL, MCDL, and WMCDL) and No Rule (LEX, WADD, MCD, and WMCD).

From this figure, we can see that for all user patterns Layer Rule produces a better result than No Rule, and the average improvement is 17.26 percent. It shows the effectiveness of applying the constraint matching rules.

Fig. 4 shows the the comparison between average MRR values from the approaches using Quantity Rule (LEXQ, WADDQ, MCDQ, and WMCDQ) and the ones using Layer Rule for all user patterns. We could see that for some user patterns, Layer Rule produces a better result, and for others, Quantity Rule produces a better result. On average, Quantity Rule is slightly better than Layer Rule with an improvement of 7.09 percent. It shows that constraint matching can improve the accuracy of the ranking algorithm, and a stricter matching rule gives a more accurate result.

4.4 Results Comparing Our Algorithm with Single Strategy based Ranking Algorithms

In this set of experiments, we again set the K value to 5. Fig. 5 shows the comparison between our algorithm and those single-strategy-based ranking algorithms for all the multi-strategy following patterns. Here the MRR value for our algorithm is calculated on the testing data.

From this figure, 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 percent 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, and thus it is necessary to consider multiple strategies. Overall speaking, the best performing algorithms are from the MCD family, either one of the WMCD or MCD algorithms.

Since many QoS-based service selection algorithms use the weighted sum (WADD) as the default decision strategy, we compare the results from our algorithm with the WADD algorithm as shown in Fig. 6. The improvement from our algorithm is obvious. The average increase on the MRR value is 54.86 percent. So if we integrate our algorithm into any existing selection system, its accuracy can be largely improved.

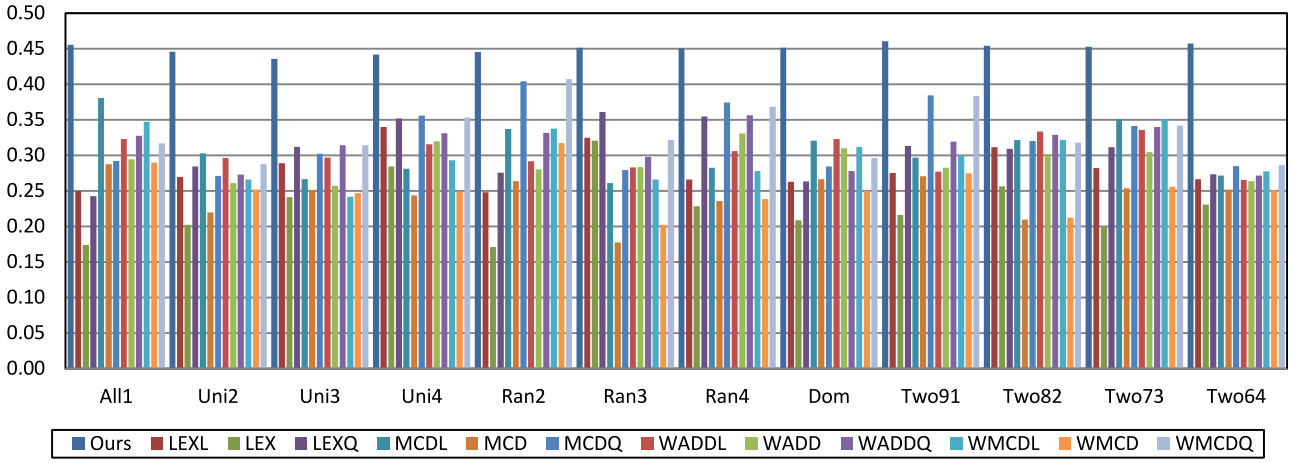


Fig. 5. Comparison between our algorithm and individual algorithms on MRR values for all user patterns.

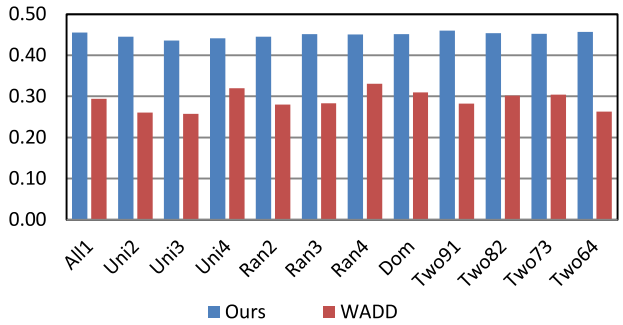


Fig. 6. Comparison between our algorithm and WADD algorithm.

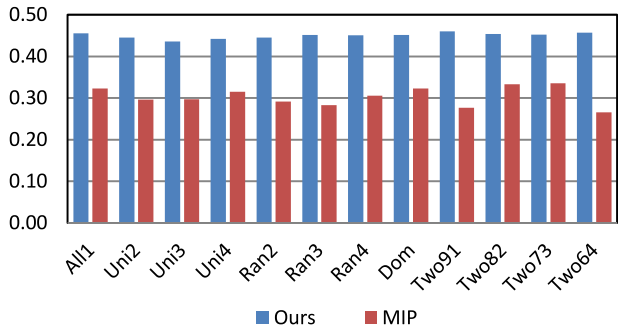


Fig. 7. Comparison between our algorithm and MIP algorithm.

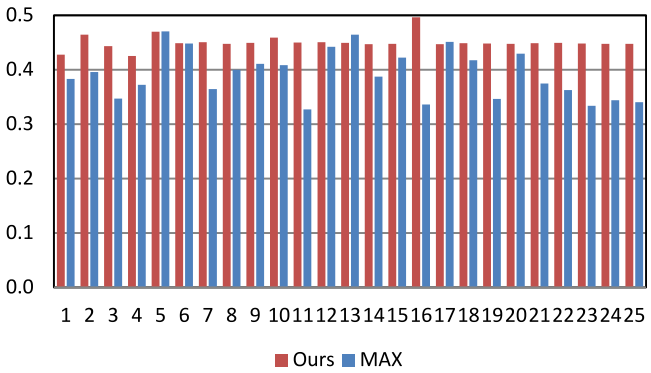


Fig. 8. Comparison between our algorithm and the best individual algorithm for 25 Dom users.

Since the way we implement our WADDL algorithm is very similar to the MIP algorithm (constraint checking first to put services into categories, and then use the weighted sum of multiple QoS values as the objective function for optimization), we consider WADDL as our implementation of the MIP algorithm. In Fig. 7, we compare the results from our algorithm with this MIP algorithm. We could see a clear advantage from our algorithm on the MRR value. The average improvement is 48.15 percent.

Our algorithm performs well not only for average users, but also for individual users when they are following multiple strategies. Fig. 8 shows the comparison between our algorithm and the best performing individual algorithms for 25 users who have dominating strategies (Dom). The best performing algorithm could be one of the 12 basic ranking algorithms, and it is different for different users. We could see that our algorithm performs consistently 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.

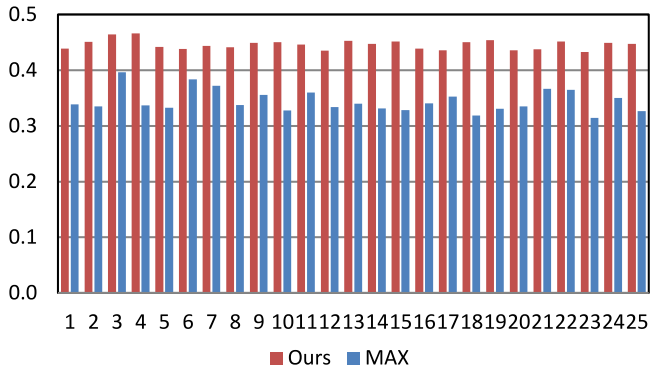


Fig. 9. Comparison between our algorithm and the best individual algorithm for 25 Uni2 users.

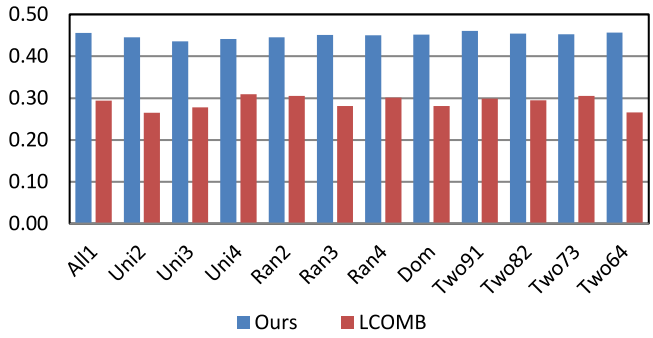


Fig. 10. Comparison of our algorithm with linear combination algorithm.

Fig. 9 shows the comparison between our algorithm and the best performing individual algorithms for 25 users who uniformly use 2 ranking strategies (Uni2). We could see that our algorithm performs consistently better than the best performing individual algorithms.

The same conclusion can also be drawn for all the other patterns when users follow multiple strategies. When users only follow one strategy all the time, the result is different, and there is no clear winner between our algorithm and the best performing individual algorithm. Since whether a user follows one strategy or multiple strategies is not known in advance when the selection system is built, and the decision-making patterns are different for different users, the advantage of using the learned model is obvious for average users.

4.5 Results Comparing Our Algorithm with the 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. Fig. 10 shows the comparison result. The K value is still 5.

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

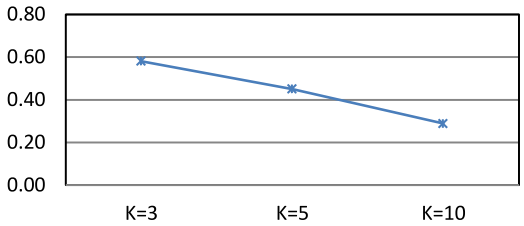


Fig. 11. Comparison of our algorithm with different *K* values.

improvement from our algorithm is 55.13 percent, which shows the effectiveness of the learning step.

4.6 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 Fig. 11.

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.

5 Conclusions

In this paper, we proposed 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 constraint matching rules. To figure out what strategies a user follows and how they are followed, a learning-to-ranking 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. Our experiment result showed the effectiveness of the proposed approach.

There are a few directions we would like to work on in the future. Currently, we used the simulated dataset to test our algorithm, which may not truly reflect the real scenarios. So the first extension to this work is to implement a fully functioning service selection system in a service registry, and then gradually collect the real data to test our algorithm. Second, we could integrate the decision strategy into some state-of-the-art selection models such as Skyline, AHP as the base selection algorithms to compare with our approach in which we use MIP-based constraint matching rules. Third, we would like to explore the possibility of using other learning algorithms or proposing a more customized learning algorithm to further improve the result. Although there have been many models used for QoS-based service selection, there is no evidence showing one model offers much better result than others, and there is no de facto model everyone agrees to use. So our last research direction is to apply the learning-to-rank algorithm on multiple QoS-based service selection algorithms (such as AHP, MIP, Skyline, vector-based) to learn an ensemble model which hopefully could provide a more optimal result than individual algorithms.

Acknowledgments

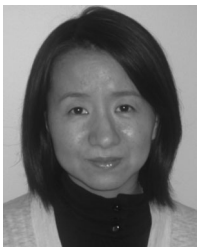
The work of Chen Ding was partially sponsored by Natural Science and Engineering Research Council of Canada (Grant 299021–2010).

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 Theory*, vol. 17, no. 8, pp. 1378–1398, 2009.
[2] K. Kritikos and D. Plexousakis, "Mixed-integer programming for QoS-based web service matchmaking," *IEEE Trans. Services Comput.*, vol. 2, no. 2, pp. 122–139, Jul. 2009.
[3] Q. Yu and A. Bouguettaya, "Computing service skyline from uncertain QoWS," *IEEE Trans. Services Comput.*, vol. 3, no. 1, pp. 16–29, Apr. 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. New York, NY, USA: Random House, 1985.
[5] J. W. Payne, J. R. Bettman, and E. J. Johnson, "Adaptive strategy selection in decision making," *J. Exp. Psychology: Learning, Memory, Cognition*, vol. 14, no. 3, pp. 534–552, 1988.
[6] T. Y. Liu, "Learning to rank for information retrieval," *J. Foundations Trends Inform. Retrieval*, vol. 3, no. 3, pp. 225–331, Mar. 2009.
[7] C. W. Hang and M. P. Singh, "From quality to utility: Adaptive service selection framework," in *Proc. Int. Conf. Service Oriented Comput.*, 2010, pp. 456–470.
[8] K. Benouaret, D. Benslimane, and A. Hadjali, "*WS-Sky*: An efficient and flexible framework for QoS-aware web service selection," in *Proc. 9th Int. Conf. Service Comput.*, 2012, pp. 146–153.
[9] W. Rong, K. Liu, and L. Liang, "Personalized web service ranking via user group combining association rule," in *Proc. 7th Int. Conf. Web Services*, 2009, pp. 445–452.
[10] Z. Zheng, Y. Zhang, and M. R. Lyu, "CloudRank: A QoS-driven component ranking framework for cloud computing," in *Proc. 29th IEEE Int. Symp. Reliable Distributed Syst.*, 2010, pp. 184–193.
[11] M. O. Shafiq, R. Alhajj, and J. Rokne, "On the social aspects of personalized ranking for web services," in *Proc. IEEE Int. Conf. High Perform. Comput. Commun.*, 2011, pp. 86–93.
[12] Q. Zhang, C. Ding, and C. H. Chi, "Collaborative filtering based service ranking using invocation histories," in *Proc. Int. Conf. Web Services*, 2011, pp. 195–202.
[13] G. Kang, J. Liu, M. Tang, X. Liu, B. Cao, and Y. Xu, "AWSR: Active web service recommendation based on usage history," in *Proc. 19th Int. Conf. Web Services*, 2012, pp. 186–193.
[14] R. Riedl, E. Brandstätter, and F. Roithmayr, "Identifying decision strategies: A process- and outcome-based classification method," *Behavior Res. Method*, vol. 40, no. 3, pp. 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*, vol. 80, no. 1–3, pp. 107–141, Aug. 1992.
[16] I. Jeffreys, "The use of compensatory and non-compensatory multi-criteria analysis for small-scale forestry," *Small-scale Forest Econom., Manag. Policy*, vol. 3, no. 1, pp. 99–117, 2004.
[17] S. Ran, "A model for web services discovery with QoS," *ACM SIGecom Exchanges*, vol. 4, no. 1, pp. 1–10, 2003.

[18] E. Al-Masri and Q. H. Mahmoud, "QoS-based discovery and ranking of web services," in *Proc. 16th Int. Conf. Comput. Commun. Netw.*, 2007, pp. 529–534.
[19] 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, J. Figueira, S. Greco, M. Ehrgott, Eds. New York, NY, USA: Springer, pp. 346–406, 2005.
[20] A. Ruiz-Cortes, O. Martin-Diaz, A. D. Toro, and M. Toro, "Improving the automatic procurement of web services using constraint programming," *Int. J. Cooperative Inform. Syst.*, vol. 14, no. 4, pp. 439–468, 2005.
[21] Q. Wang, H. Wang, Y. Li, G. Xie, and F. Liu, "A semantic QoS-aware discovery framework for web services," in *Proc. IEEE Int. Conf. Web Services*, 2008, pp. 129–136.
[22] Y. Koren. (2014, Jan.). The bellkor solution to the netflix grand prize. [Online]. Available: http:// netflixprize.com/assets/ GrandPrize2009_BPC_BellKor.pdf
[23] J. Xu and H. Li, "AdaRank: A boosting algorithm for information retrieval," in *Proc. 30th Annu. Int. ACM SIGIR Conf. Res. Develop. Inform. Retrieval*, 2007, pp. 391–398.
[24] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An ef cient boosting algorithm for combining preferences," *J. Machine Learning Res.*, vol. 4, pp. 933–969, 2003
[25] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press, 2009.
[26] R. Karim, C. Ding, and C. H. Chi, "Multiple non-functional criteria service selection based on user preferences and decision strategies," in *Proc. IEEE Int. Conf. Service-Oriented Comput. Appl.*, 2011, pp. 1–8.

Muhammad Suleman Saleem received the MSc degree in computer science from Ryerson University, Toronto, ON, Canada, in 2014. He is currently working in SK Wireless as a System Analyst. His research interests include QoS-based web service selection, decision theory, and learning to rank algorithms. He has one paper published in IEEE International Conference on Web Services (ICWS).



Chen Ding received the PhD degree in computer science from National University of Singapore, Singapore, in 2003. She is currently an Associate Professor in the Department of Computer Science at Ryerson University, Toronto, ON, Canada. Her research interests include web service selection, recommender systems, social data analytics, web mining, behavior analytics, and web search. She has publications in various international journals and conferences such as *Future Generation Computer Systems* (FGCS), *Service Oriented Computing and Applications* (SOCA), *International Journal of Web Service Research* (JWSR), *IEEE International Conference on Web Services* (ICWS), IEEE International Conference on Service Computing (SCC), International Conference on Service Oriented Computing (ICSOC), etc. She also serves various international conferences as a program committee member.

Xumin Liu received the PhD degree in computer science from Virginia Tech, Blacksburg, VA. She is currently an Assistant Professor in the Department of Computer Science at Rochester Institute of Technology, New York, NY. Her research interests include data management, data mining, Semantic Web, service computing, social computing, and cloud computing. Her research has been published in various international journals and conferences, such as the *International Journal on Very Large Data Bases* (VLDB journal), *IEEE Transactions on Service Computing* (TSC), *International Journal of Web Service Research* (JWSR), *IEEE International Conference on Web Services* (ICWS), International Conference on Collaborative Computing: Networking, Applications, and Worksharing (CollaborateCom), etc. She frequently serves as a program committee member for various international conferences. She is also a reviewer for various journals, including ACM transactions and IEEE transactions, and others.

Chi-Hung Chi received the PhD degree from Purdue University, West Lafayette, IN. He is currently a science leader in the Digital Productivity and Services Flagship in CSIRO, Clayton South, Australia. Before joining CSIRO in 2012, he has worked in industry (Philips Research and IBM) and universities (Chinese University of Hong Kong, National University of Singapore, and Tsinghua University) for more than 20 years. His current research areas include behavior informatics, big data and analytics, service engineering, cloud computing, and social networking. He has published more than 240 papers in international conferences and journals and holds 6 U.S patents. He was the program/general chairman of WCW 2004, AWCC 2004, IEEE SOSE 2006, ICSOC 2009, SCC 2009, SIE 2010, EDOC 2011, EDOC 2012, and CWI 2012 and PC members of about 100 conferences. He is currently on advisory board of European Research Institute on Service Science (ERISS), Steering Committee Member of IEEE International Enterprise Distributed Object Computing Conference (EDOC), an Associate Editor of International Journal of Big Data, on editorial advisory board of 6 international journals. The technologies he developed have also been transferred into industry.