AN EFFICIENT SIMILARITY-BASED MODEL FOR WEB SERVICE RECOMMENDATION

Abdullah Abdullah\textsuperscript{1,2}, Xining Li\textsuperscript{1}

School of Computer Science
\textsuperscript{1}University of Guelph, Guelph, ON, Canada
\textsuperscript{2}University of Mosul, Mosul, Iraq
{aabdulla, xli}@uoguelph.ca

Abstract

With the emergence of Cloud and Mobile Computing paradigms which adopt web services (WS) as a means of management, companies worldwide are actively deploying web services within their business environments. Consequently, designing effective Web service recommendation mechanisms is receiving more research attention. Traditional Neighborhood-based Collaborative Filtering (CF) models fail to capture the actual relationships between users and services due to data sparsity. In contrast, Random Walk (RW) algorithm, which is categorized as a sparsity-tolerant recommendation approach, suffers from poor recommendation accuracy. In this paper, we first propose an Integrated-Model QoS-based Graph (IMQG), in which users and services represent nodes while weighted Quality of Service (QoS) magnitudes and User/Service similarity measurements serve as edges. Variants of Jaccard coefficient are used to separately compute these similarities. Then, Top-k Random Walk algorithm is applied to generate a final recommendation list. Further, we reduce the model by selecting only a subset of users to better guide RW algorithm. Comprehensive experiments are conducted on a real-world dataset where analysis of results shows high improvement in recommendation accuracy with more tolerance to data sparsity. Utilizing the improved model, a considerable reduction in computation time is achieved while maintaining the recommendation accuracy.

Keywords: Web service; Recommender Systems, QoS, Jaccard coefficient; Random Walk;

1 INTRODUCTION

Service-Oriented Computation (SOC) has been recently emerged as a promising Web paradigm in different application fields such as e-commerce, enterprise application integration, etc. About 30000 Web services are available on the Internet from around 8000 providers, as of June 2013 (Yao et al., 2013). With these growing numbers, assisting service users to select appropriate Web services that fulfill their needs becomes difficult and time-consuming. Hence, the need for reliable Web service recommender systems (RS) is attracting more researchers to design effective and applicable recommendation models.

Web service RSs are primarily divided into two types, functional (e.g., name, input and output) and non-functional (i.e., Quality of Service) based systems. Qualities of Service attributes include Response Time (RT), Failure Rate (FR), Throughput, Availability, etc. Most prior works in QoS-based WS recommendation utilizes neighborhood-based Collaborative Filtering (Jiang et al., 2011; Jin, Chai & Si, 2004; Zheng et al., 2009). In which similarities among users/services are computed to predict missing QoS values to an active user by considering a set of nearest similar users/services. Although the CF approach is currently considered the most common approach, it suffers from real recommendation issues such as data sparsity.

In recent literature, Random Walk algorithm has been successfully used in recommendation models to alleviate data sparsity (Zhang et al., 2013; Zhou et al., 2013). Yet, it still suffers from poor performance in terms of recommendation accuracy, especially when applied on a traditional user-item recommendation model. Taking the advantage of RW into consideration, we were motivated to design an effective recommendation model which utilizes Random Walk algorithm as an underlying approach. By effectively capturing and integrating actual relationships among users and services into our model, recommendation accuracy is improved compared to traditional Random Walk based model. The key contributions of this work can be summarized as in the following:

1. We present Jaccard coefficient in several variants, appropriate for WS recommendation.
2. We propose an effective Integrated-Model QoS-based Graph model for WS recommendation, in which User-Service Bipartite Graph is fused with User-based and Service-based Unipartite Graphs.
3. We present an efficient recommendation model, in which RW algorithm is better guided using a selected subset of Jaccard similarities, instead of the entire similarity set. The goal is to minimize computation time, required in the conventional RW algorithm, while maintaining close levels of recommendation accuracy.
4. We conduct a set of extensive experiments on a real-world WS dataset to validate our recommendation framework. Comprehensive analysis on the impact of various experimental parameters is also provided.
Results show that improved recommendation accuracy can be obtained by utilizing the proposed model in different integration approaches.

The rest of this paper\(^1\) is organized as follows. First, Section 2 discusses the related work. Section 3 introduces the proposed WS recommendation framework, including User-based and Item-based similarity graphs. In section 4, we describe how the Integrated-Model QoS-based Graph is constructed. We show the evaluation methodology in section 5. Section 6 and 7 describe the Efficient Recommendation Model and its evaluation, respectively. Finally, we conclude in section 8.

2 RELATED WORK

Collaborative Filtering approaches are widely used techniques for personalized QoS prediction based on past QoS data. They can be divided into two categories, neighborhood-based and model-based.

Neighborhood-based CF algorithms can be further divided into three types, User-based (Jin, Chai & Si, 2004), Item-based (Deshpande & Karypis, 2004) and hybrid (Jiang et al., 2011; Zheng et al., 2009; Zheng et al., 2011; Chen et al., 2014) approaches. In User-based approaches, ratings for active users are predicted based on the ratings of their similar users; while in Item-based approaches, the new ratings are estimated based on the items similar to those selected by active users. The hybrid approach is a combination of the former two approaches, thus it can achieve higher prediction accuracy. In contrast, Model-based CF is a training based approach which utilizes portion of the data to train a predefined model, e.g., clustering models (Tsai & Hung, 2012), aspect models (Hofmann, 2004) and latent factor models (Canny, 2002), and then employs the trained model for QoS prediction.

In general, Neighborhood-based CF is preferred over model-based CF since it is more intuitive to interpret QoS prediction results. A clear drawback of neighborhood-based CF is its vulnerability to data sparsity. Particularly, this approach cannot automatically recognize indirect similar neighbors for a given user.

Therefore, to alleviate data sparsity issue, some researchers adopt RW algorithm for recommendation tasks. Yildirim & Krishnamoorthy et al. (2008) applied the random walk to an item similarity graph, in which items represent nodes and item similarities serve as edges. It assigns each item a stationary visiting probability as a ranking score and then finds the higher ranked items. In (Zhou et al., 2013; Zhang et al., 2013), random walk is carried on a hybrid graph, which is extracted by treating both users and items as nodes and user-item interactions as edges. Similarly, items with higher stationary visiting probabilities are more likely to be recommended to target users. The authors in (Hu, Peng & Hu, 2014) proposed a QoS recommendation approach based on Random Walk method. They integrated temporal information into both similarity measurement and QoS prediction of the traditional CF. Then a hybrid personalized random walk algorithm is applied to the user and service graphs to find more indirect similar users and services. Finally, prediction results of both user and service random walks are combined to form final QoS predictions.

The works based on random walk do not incorporate similarity measures separately within a comprehensive recommendation model. In fact, an effective Web service recommendation approach needs to infer sound similarities among users/items, rather than to rely only on direct user ratings to items.

However, to the best of our knowledge, none of the existing methods for Web service recommendation takes into account the similarity of Users and Services separately and then integrate them with previously gathered QoS information into a comprehensive recommendation model. Furthermore, the model is reduced by filtering out unlikely supportive similarities resulting into a significant reduction in the amount of information and time consumption. By applying Random Walk algorithm on the Integrated Model, higher recommendation accuracy is obtained with great tolerance to data sparsity.

3 WS RECOMMENDATION FRAMEWORK

In Figure 1, we illustrate the main stages of the proposed recommendation framework, which are briefly described as follows:

a) Service users contribute their experienced QoS data to a central log.

b) User-Service Bipartite graph is created, containing weighted QoS values.

c) Using the Jaccard coefficient, similarity magnitudes among users are computed to build User-based Unipartite graph.

d) Similarity values among services are also computed using the same coefficient to build Service-based Unipartite graph.

e) User-Service Bipartite graph with User-based and Service-based Unipartite graphs are fused to create the IMQG model.

f) Top-k Random Walk algorithm is applied to the IMQG model to generate a list of best & Web services based on their relevance scores, while ignoring services that the user already invoked.

In this work, we adopt a graph structure as a building block of our framework. Thus, we create three types of graph, i.e., Unipartite, Bipartite, and Integrated-Model graphs. We will use the terms “efficient model” and “reduced model” interchangeably throughout this paper.

\(^1\)An early version of this paper was published at the IEEE 22nd International Conference on Web Services (ICWS-2015) (Abdullah & Li, 2015).
3.1 User-Service Bipartite Graph

The first graph in the framework is User-Service Bipartite Graph, which is illustrated in Figure 2 and defined as follows:

**Definition 1.** (User-Service Bipartite Graph). A User-Service Bipartite Graph is defined as an undirected graph $G = (V, E)$, where $V$ and $E$ represent finite sets of nodes and edges respectively. Set $V$ is a union of two disjoint subsets $V = V_1 \cup V_2$, where $V_1$ consists of user nodes and $V_2$ contains service nodes. Every edge has the form $e = (a, b, w)$, where $a \in V_1$, $b \in V_2$, and $w$ is a weighted QoS value.

Table 1 shows a sample graph that is realized as a weighted adjacency matrix $W_{QoS}$, where $s_1 \cdots s_5$ are service items and $u_1 \cdots u_3$ are service users. The $\times$ cell denotes absence of the QoS value.

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$s_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>$w_{1,1}$</td>
<td>$w_{1,2}$</td>
<td>$w_{1,3}$</td>
<td>$\times$</td>
<td>$w_{1,5}$</td>
</tr>
<tr>
<td>$u_2$</td>
<td>$\times$</td>
<td>$w_{2,2}$</td>
<td>$\times$</td>
<td>$w_{2,4}$</td>
<td>$w_{2,5}$</td>
</tr>
<tr>
<td>$u_3$</td>
<td>$\times$</td>
<td>$w_{3,2}$</td>
<td>$w_{3,3}$</td>
<td>$w_{3,4}$</td>
<td>$w_{3,5}$</td>
</tr>
</tbody>
</table>

3.2 Similarity Graphs

A key advantage of using graph structure is that its local structural characteristics can be effectively exploited within recommendation techniques. In particular, several similarity forms among graph nodes can be utilized to recommend unknown nodes to active users. In this work, we employ a widely used similarity coefficient (i.e., Jaccard coefficient) to compute similarity magnitudes of every two users and services. In the following, we first define similarity graphs and then present various forms of Jaccard coefficient used in constructing these graphs.

3.3 Jaccard Similarity for Recommendation

Jaccard similarity coefficient is defined as the size (cardinality) of intersection divided by the size of the union of two sets (Jaccard, 1901). Given that $E_{ui}$ and $E_{uj}$ are two rating sets of users $u_i$ and $u_j$ respectively, Jaccard coefficient can be defined as follows:

$$Jc(E_{ui}, E_{uj}) = \frac{|E_{ui} \cap E_{uj}|}{|E_{ui} \cup E_{uj}|}$$ (1)

Although the definition is straightforward, its implementation can considerably vary, resulting into various similarity outcomes. In fact, there are different approaches to compute the union and intersection of two rating sets based on the method of handling positive and negative (i.e., liked and disliked) ratings. For instance, a simple approach evenly treats liked and disliked common items. This is the most common approach in current collaborative Recommender Systems. Another possible approach is to differentiate between the two types such that greater weight is assigned to liked items over disliked ones. Other ways also consider the common items which are unrated on one side or have opposite ratings to each other.
Table 2. Running example: $W_{QoS}$ with binary rating scale

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$s_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\times$</td>
<td>1</td>
</tr>
<tr>
<td>$u_2$</td>
<td>$\times$</td>
<td>0</td>
<td>$\times$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$u_3$</td>
<td>$\times$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Unlike with binary rating scale, illustrated in Table 2, computing Jaccard coefficient is more complicated with n-ary rating scale (e.g., QoS values). Thus, we first need to divide the n-ary rating scale into three sets, good, bad and unrated. Let $G$, $B$ and $Z$ be the sets of good, bad and unrated, respectively.

\[
G = \{ x : x > T_{QoS} \} \\
B = \{ x : x \leq T_{QoS} \} \\
Z = \{ x : x \notin G \land x \notin B \}
\]

where $T_{QoS}$ is a predefined QoS threshold value. Accordingly, we define:

\[
P_{ij} = \{ s_k : w_{ijk} \in G \land w_{ijk} \in G, i \neq j \}, \quad p_{ij} = |P_{ij}| \quad (5) \\
N_{ij} = \{ s_k : w_{ijk} \in B \land w_{ijk} \in B, i \neq j \}, \quad n_{ij} = |N_{ij}| \quad (6) \\
D_{ij} = \{ s_k : (w_{ijk} \in G \land w_{ijk} \in B) \lor (w_{ijk} \in B \land w_{ijk} \in G), i \neq j \}, \quad d_{ij} = |D_{ij}| \quad (7) \\
U_{ij} = \{ s_k : (w_{ijk} \in Z \land w_{ijk} \in Z) \lor (w_{ijk} \notin Z \land w_{ijk} \notin Z), i \neq j \}, \quad u_{ij} = |U_{ij}| \quad (8)
\]

where $p_{ij}$ and $n_{ij}$ denote the number of positive and negative ratings, respectively, in both sets $E_{u_i}$ and $E_{u_j}$; $d_{ij}$ is the number of positive ratings in set $E_{u_i}$ but negative in set $E_{u_j}$ or vice versa; and $u_{ij}$ is the number of unrated items in set $E_{u_i}$ but not set $E_{u_j}$ or vice versa. Based on the above definitions, we present four different forms to compute Jaccard similarity coefficient, namely, $Jc_1, Jc_2, Jc_3$ and $Jc_4$. The simple form of Jaccard coefficient is defined as:

\[
Jc_1 \left( E_{u_i}, E_{u_j} \right) = \frac{p_{ij} + d_{ij} + n_{ij}}{p_{ij} + d_{ij} + n_{ij} + u_{ij}} \quad (9)
\]

This definition clearly ignores the quality of rating by uniformly considering all rated items, whether they are positive or negative. In particular, it computes the ratio of the number of rated common items to the number of all items except the completely-unrated common ones. In other words, only the items, which have no rating of the two users, are excluded. Table 3 shows user similarity matrix of our running example based on $Jc_1$.

As the above definition does not differentiate between positive and negative ratings, the second form of Jaccard coefficient puts more emphasis on positive ratings. It is defined as:

\[
Jc_2 \left( E_{u_i}, E_{u_j} \right) = \frac{p_{ij}}{p_{ij} + d_{ij} + n_{ij} + u_{ij}} \quad (10)
\]

Obviously, this definition computes the ratio of the number of positively-rated common items to the number of all items except the completely-unrated common ones. Table 4 shows user similarity matrix of our running example based on $Jc_2$.

While the previous two definitions take into account negative and partially-unrated common items to some extent, the third form of Jaccard coefficient entirely avoids them. It is defined as:

\[
Jc_3 \left( E_{u_i}, E_{u_j} \right) = \frac{p_{ij}}{p_{ij} + d_{ij}} \quad (11)
\]

In this definition, the ratio of the number of positively-rated common items to the number of positively and differently-rated common ones is computed. Table 5 shows the user similarity matrix of our running example using $Jc_3$.

Although $Jc_3$ form ignores partially-unrated common items, it essentially focuses on how many positively-rated items exist. However, the last variation of Jaccard coefficient includes also negatively-rated items. In this form, two users are considered similar when their ratings are matched, regardless positively or negatively. The emphasis

---

Table 2. Running example: $W_{QoS}$ with binary rating scale

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_4$</th>
<th>$s_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\times$</td>
<td>1</td>
</tr>
<tr>
<td>$u_2$</td>
<td>$\times$</td>
<td>0</td>
<td>$\times$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$u_3$</td>
<td>$\times$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3. $Jc_1$ user similarity matrix

<table>
<thead>
<tr>
<th></th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1.00</td>
<td>0.40</td>
<td>0.60</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0.40</td>
<td>1.00</td>
<td>0.75</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0.60</td>
<td>0.75</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 4. $Jc_2$ user similarity matrix

<table>
<thead>
<tr>
<th></th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1.00</td>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0.20</td>
<td>1.00</td>
<td>0.25</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0.20</td>
<td>0.25</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 5. $Jc_3$ user similarity matrix

<table>
<thead>
<tr>
<th></th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>1.00</td>
<td>0.50</td>
<td>0.33</td>
</tr>
<tr>
<td>$u_2$</td>
<td>0.50</td>
<td>1.00</td>
<td>0.50</td>
</tr>
<tr>
<td>$u_3$</td>
<td>0.33</td>
<td>0.50</td>
<td>1.00</td>
</tr>
</tbody>
</table>
here is mainly on how similar are the two users in their liked and disliked items. It is defined as:

\[
Jc_4(E_{ui}, E_{uj}) = \frac{p_{ui} + n_{ij}}{p_{ui} + d_{ij} + n_{ij}}
\]  (12)

In this definition, the ratio of the number of positively or negatively-rated common items to the number of positively, negatively and differently-rated common ones is calculated. Table 6 shows the user similarity matrix of our running example based on Jc4.

Table 6. Jc₄ user similarity matrix

<table>
<thead>
<tr>
<th></th>
<th>u₁</th>
<th>u₂</th>
<th>u₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>u₁</td>
<td>1.00</td>
<td>0.50</td>
<td>0.33</td>
</tr>
<tr>
<td>u₂</td>
<td>0.50</td>
<td>1.00</td>
<td>0.66</td>
</tr>
<tr>
<td>u₃</td>
<td>0.33</td>
<td>0.66</td>
<td>1.00</td>
</tr>
</tbody>
</table>

From the above definitions, it is obvious that measuring similarity among users using the Jaccard coefficient can be significantly varied depending on the different weights given to positive, negative, and unrated items when computing the final similarity magnitude. Such flexibility and simplicity add to the advantages of adopting Jaccard coefficient as a similarity measure in recommendation models.

Although the above definitions are centered on computing the similarities among users, we can easily use them for calculating the similarities among services by replacing users’ rating sets with services’ rating sets.

For the value of \( T_{QoS} \), we think that it must be carefully chosen by the system administrator. We also think that it is possible to formulate a method for this selection, yet it is out of the scope of this work.

In our framework, we evaluate the four different forms of Jaccard coefficient, which are used to construct User-based and Service-based similarity graphs. Initially, our expectation was that \( Jc_3 \) and \( Jc_2 \) will outperform the rest due to their great emphasis on positive rating, which naturally reflects the similarity of users’ tastes.

4 Integrated-Model QoS-Based Graph (IMQG)

Once the User-Service graph and both similarity graphs are created, we fuse them into one graph called the Integrated-Model QoS-based Graph which is defined as follows:

Definition 4 (Integrated-Model QoS-based Graph). An Integrated-Model QoS-based Graph is defined as an undirected graph \( G = (V, E) \), where \( V \) and \( E \) represent finite sets of nodes and edges respectively. Set \( V \) is obtained as a union of two subsets \( V = V_1 \cup V_2 \), where \( V_1 \) consists of user nodes and \( V_2 \) contains service nodes. Every edge has the form \( e = (x, y, w) \) where \( x \in V \) and \( y \in V \) or vise versa, then \( w \) is a weighted QoS value, otherwise it is a similarity value between \( x \) and \( y \).

As a multi-partite graph, the IMQG model can be viewed as a transition probability matrix. Random Walk over a multi-partite graph is a multi-step approach that simulates a navigating process from one part of the graph to another, e.g., from a set of user nodes to the corresponding set of item nodes. It is a stochastic process in which the initial state is known, while the next state \( S \) is provided by a probability distribution. The distribution is represented by constructing the transition probability matrix \( A \), where the value of \( A_{i,j} \) is the probability of moving from node \( i \) (at time \( n \)) to node \( j \) (at time \( n+1 \)) as in the following:

\[
A_{ij} = P(S_{n+1} = j | S_n = i)
\]  (13)

Figure 3 illustrates how the IMQG model is created as a transition probability matrix. A complete transition matrix is initially created from a number of sub-matrices, namely, User-User, User-Service, Transposition of User-Service and Service-Service sub-matrices. These sub-matrices also have different types of weight. Particularly, User-Service sub-matrix contains weighted QoS values, while User-User and Service-Service sub-matrices store similarity magnitudes. In order to conform to Random Walk algorithm, the next step is to row-normalize the sub-matrices.

If a similarity sub-matrix is excluded from the recommendation process, it is substituted by a self-transition sub-matrix to allow the walk to stay in place. For example, when applying Random Walk with only service-service similarity sub-matrix, user-user similarity sub-matrix is replaced by user identity matrix, i.e., with ones on the main diagonal and zeros elsewhere. To control the effect of similarity or self-transition feature, a certain probability \( \alpha \) is applied. Additionally, User-Service and its Transposition sub-matrix are row-normalized by the factor \( \beta = 1 - \alpha \).

The following is a pseudo code of Top-k RW, in which the initial state query vector \( v_0 \) indicates an active user. This user is selected to start the walk by setting its corresponding element in \( v_0 \) to 1, while all remaining elements (i.e., all other users) are initialized to 0.
Algorithm 1. Topk Random Walk over IMQG

Input:
\( A_{n,n} \), a transition probability matrix (IMQG).
\( v_0 \), an initial state query vector of size \( n \).
\( N \), number of steps.
\( k \), number of top recommendation items required.

Output:
\( v_N \), a final state vector, i.e., \( v_0 \) at step \( N \).

Procedure:
1: \( t = 1 \)
2: repeat
3: \( \text{//Walk one step ahead} \)
4: \( v_0 = A_{n,n} \times v_0 \)
5: \( t = t + 1 \)
6: until \((t == N)\)
7: Remove user already rated service items.
8: Sort \( v_N \) in descending order.
9: Select Top \( k \) service items.

5 Evaluation And Experiments

5.1 Dataset

We conduct a series of experiments based on a real QoS dataset (Zheng et al., 2011). The dataset is distributed in 150 files, each of which represents a service user’s invocation record. In each file, there are about 10,000 Web service invocations on 100 publicly available Web services. In total, the dataset contains more than 1.5 million invocations.

First, we create two \( 150 \times 100 \) matrices, i.e., Response Time and Failure Rate matrices. Given that service user \( u \) has invoked service item \( i \) about 100 times, we calculate Response Time rating as the mean Response Time of all invocations. Table 7 shows a sample part of RTT matrix. In contrast, Failure Rate rating is computed as the ratio of number of failures to the total number of invocations. To conform to RW, one is added to shift all zero entries. Table 8 shows a sample part of FR matrix. Finally, we assign both ratings to the corresponding entries in RT and FR matrices respectively.

Table 7. RTT matrix

<table>
<thead>
<tr>
<th></th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>...</th>
<th>( S_{100} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u_1 )</td>
<td>80</td>
<td>3846</td>
<td>...</td>
<td>4562</td>
</tr>
<tr>
<td>( u_1 )</td>
<td>586</td>
<td>458</td>
<td>...</td>
<td>64</td>
</tr>
<tr>
<td>( u_3 )</td>
<td>61</td>
<td>720</td>
<td>...</td>
<td>1520</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( u_{150} )</td>
<td>245</td>
<td>92</td>
<td>...</td>
<td>43</td>
</tr>
</tbody>
</table>

Table 8. FR matrix

<table>
<thead>
<tr>
<th></th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>...</th>
<th>( S_{100} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( u_1 )</td>
<td>1.0000</td>
<td>1.0000</td>
<td>...</td>
<td>1.0000</td>
</tr>
<tr>
<td>( u_1 )</td>
<td>1.0092</td>
<td>1.0102</td>
<td>...</td>
<td>1.0000</td>
</tr>
<tr>
<td>( u_3 )</td>
<td>1.0000</td>
<td>1.0170</td>
<td>...</td>
<td>1.0000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( u_{150} )</td>
<td>1.0095</td>
<td>1.0107</td>
<td>...</td>
<td>1.0020</td>
</tr>
</tbody>
</table>

5.2 Testing Methodology

In this work, we adopt a testing methodology similar to the one, described in (Cremonesi, Koren & Turrin, 2010). For each QoS attribute, i.e., RT and FR, the set of service users \( U \) is split into two subsets: training set \( U_N \) and test set \( U_T \). As a result, four sub-matrices are created, namely, \( RT_{U_N} \), \( RT_{U_T} \), \( FR_{U_N} \) and \( FR_{U_T} \). To simulate real-world sparse data, we randomly remove entries (i.e., QoS values) from training matrices with different densities (i.e., 10%, 20%, etc.). Similarly, density of test matrices is also lowered to 50% to provide a necessary set of unrated service items required for later evaluation process.

For each test user, we also remove a number of entries. The number of remaining entries is called a Given Number which represents the number of QoS values provided by a test user. The removed entries are the top-rated ones to ensure that their corresponding Web services are reasonably relevant to the respective test user. All removed entries for all test users are stored in set \( T \). Finally, original values of the removed entries are used to estimate the prediction accuracy.

To measure the prediction accuracy of the Top-k recommendation approach, \( \text{recall}@k \) and \( \text{precision}@k \) metrics are used. For each test service item \( i \) of test user \( u \):

1. We randomly choose one third of the service population (i.e., 33 items), unrated by user \( u \), assuming that most of them are not of interest to him.
2. We predict the ratings for test service item \( i \) and for the additional chosen service items.
3. We form a ranked list by ordering all the 34 service items based on their predicted ratings. Let \( q \) denote the rank of the test service item \( i \) within the list. The best result is where \( q \) precedes all other items, i.e., \( q = 1 \).
4. We form a top-k recommendation list by selecting the \( k \) top-ranked service items from the list. If \( q \leq k \), then it is a hit (i.e., the test service item \( i \) is recommended to user \( u \)), otherwise it is a miss. Probability of hit increases with \( k \). When \( k = 34 \) there is always a hit.

Intuitively, recall of a test service item can be either 0 (in case of a miss) or 1 (in case of a hit). Similarly, precision can be either 0 or \( \frac{1}{k} \). Consequently, the overall \( \text{recall}@k \) and \( \text{precision}@k \) are defined by averaging over all test cases:

\[
\text{recall}(k) = \frac{\# \text{hit}}{|T|}
\]

\[
\text{precision}(k) = \frac{\# \text{hit}}{|T| \cdot k}
\]

where \( |T| \) is the total number of test entries of all test users, while \( k \) is the required size of recommendation list.
To study the effectiveness of incorporating similarity computation into WS recommendation, we present three different recommendation approaches, which can be classified under two main categories. The first includes two partially integrated models, namely, RW with User-User similarity \((RWUUJc)\) and RW with Service-Service similarity \((RWSSJc)\), while the second is a fully integrated model, i.e., RW with both Service-Service and User-User similarity \((RWUSJc)\), such that:

- \(RWUUJc = \text{User-Service Bipartite Graph} + \text{User-User Unipartite Graph}\).
- \(RWSSJc = \text{User-Service Bipartite Graph} + \text{Service-Service Unipartite Graph}\).
- \(RWUSJc = \text{User-Service Bipartite Graph} + \text{Service-Service Unipartite Graph} + \text{User-User Unipartite Graph}\).

Similarity of each approach is applied using four forms of Jaccard coefficient resulting into twelve recommendation approaches (i.e., \(RWUUJc_1\), \(RWUUJc_2\), \(RWUSJc_1\), \(RWUSJc_2\), \(RWUUJc_3\), \(RWUUJc_4\), \(RWUSJc_3\), \(RWUSJc_4\), \(RWUUJc_5\), \(RWSSJc_3\), \(RWSSJc_4\), \(RWUSJc_5\)).

Since Response Time and Failure Rate are both classified as negative QoS attributes (i.e., the lower the attribute value is, the better the quality is), we carefully take that into account in two cases. The first is when we split the n-ary QoS scale into positive and negative sub-scales in order to compute Jaccard coefficient; while the second is when we pick the Top-k Web services from the whole recommendation list (i.e., at the last step of the Top-k RW algorithm). For comparison, we adopt the basic Random Walk as a baseline approach to show to what extent the proposed IMQG model may outperform RW in terms of recommendation accuracy and sparsity tolerance. We also empirically set \(T_{QoS} = \text{median}(QoS)\) as a fair value, number of RW steps \(N = 3\) and \(\alpha = 0.2\) in all experiments.

### 5.3 Recommendation Accuracy

Table 9 and Table 10 show the recall results for Response Time and Failure Rate attributes respectively. Different recommendation approaches are subsequently examined, employing 10%, 20% and 30% density of RT and FR training matrices. While the training user number is set to 100, given numbers are varied from 10 to 30 (i.e., \(G10, G20\) and \(G30\)). Top-K is also fixed at 10. Each experiment is executed 50 times and mean recall values are reported. Experimental results of Table 9 and Table 10 show that:

1. Recommendation approaches based on \(Jc_3\) and \(Jc_4\) outperform \(Jc_4\) based approach, while \(Jc_1\) comes at the end of the list. In fact, \(Jc_1\) performs as poorly as basic Random Walk. The reason is that \(Jc_1\) uniformly treats positive and negative ratings, while \(Jc_3\) and \(Jc_4\) mainly focus on positive ratings. However, \(Jc_3\) based approach is the winner in most cases. Therefore, designing effective similarity measures is critical to accurate WS recommendations.

2. Service-based and User-Service-based integrated approaches (i.e., \(RWSSJc_3\) and \(RWUSJc_5\)) mostly produce better accuracy results than User-based approaches (i.e., \(RWUUJc_5\)).

In fact, this result indicates that Item-based similarity approaches have higher accuracy performance than User-based ones. This can be justified as follows. Given that the final recommendation is a list of services (i.e., not of users) then the role of incorporating service-to-service similarity is significant. For instance, if two services are favorable to the majority of users, their corresponding similarity value will accordingly be high. Thus, if a user has already invoked one of them, then it is highly likely that the second service will be recommended to him due to such a high similarity, in response to his recommendation query.

3. The \(Jc_4\) based approaches have some exceptional behaviors. For example, its recall results of the Failure Rate attribute are the lowest in most cases (even lower than the basic RW). Also it is noticeable that \(RWUSJc_4\) performs the worst compared to \((RWUUJc_5)\), while \(RWUUJc_3\) performs the best compared to \((RWSSJc_3\) and \(RWUSJc_3\) for the other approaches respectively. This indicates that taking negative common ratings into account when generating Jaccard coefficient has a negative effect on the overall accuracy. This is reasonable since negatively rated items are obviously of no interest to the user.

4. Competition between Service-based and User-Service-based approaches is evident. However, the former tends to perform slightly better in low densities and given numbers.

5. With the increase of training matrices density, the recommendation accuracy is significantly improved, since denser training matrices offer more information for producing high quality recommendations.

6. By increasing the given number from 10 to 30, recall values are enhanced. This observation indicates that users can receive more accurate recommendations when more QoS information is provided by service users.

### 5.4 Impact of Experimental Parameters

In the following, we study the impact of different experimental parameters on the accuracy performance of the proposed model:

#### 5.4.1 Top-k

To study the impact of the Top-k parameter on the accuracy of the Service-based recommendation approach, we vary the value of Top-k from 2 to 20 with a step value of 2. Figure 4(a) and 4(b) report recall and precision values of RT matrix under the experimental settings of training matrix.
density = 10%, given number = 20, and training user number = 100. Based on the same settings, Figure 4(c) and 4(d) show recall and precision results of FR matrix.

Figure 4 shows that the recall consistently increases with Top-k value. Additionally, recall and precision results of RWSSJc3 and RWSSJc2 steadily outperform the results of RWSSJc1 and RW. Similar to the prior experiment, we notice that RWSSJc3 comes in the middle for RT attribute while at the end for FR. In general, these observations acknowledge the positive influence of Top-k parameter on the accuracy performance.

However, in most real-world recommender systems, this parameter is mostly delimited by 20 since users usually do not look beyond 20 items in recommendation lists.

### 5.4.2 Given Number

In this experiment, we analyze the impact of the given number on the recommendation results. First, we change the given number from 5 to 40 with a step value of 5. Other experimental parameters are as follows: training user number = 100 and 130, Top-k = 10, training matrix density = 10% and 20%.

#### Table 9. Accuracy Performance Comparison (Response Time)

<table>
<thead>
<tr>
<th>Methods</th>
<th>G10 Density = 10% G20 G30</th>
<th>G10 Density = 20% G20 G30</th>
<th>G10 Density = 30% G20 G30</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>28.16% 27.61% 26.66%</td>
<td>30.55% 30.45% 30.38%</td>
<td>31.95% 32.92% 33.75%</td>
</tr>
<tr>
<td>RWUJC1</td>
<td>28.12% 28.25% 27.39%</td>
<td>28.73% 30.67% 31.51%</td>
<td>30.22% 29.99% 33.24%</td>
</tr>
<tr>
<td>RWSSJc1</td>
<td>29.20% 27.63% 28.60%</td>
<td>29.43% 28.43% 31.31%</td>
<td>30.05% 30.81% 30.96%</td>
</tr>
<tr>
<td>RWUSJc1</td>
<td>29.63% 29.76% 30.15%</td>
<td>31.14% 29.44% 30.35%</td>
<td>28.90% 29.95% 28.07%</td>
</tr>
<tr>
<td>RWUJC2</td>
<td>28.91% 28.35% 28.29%</td>
<td>31.33% 29.78% 30.44%</td>
<td>29.08% 29.83% 29.27%</td>
</tr>
<tr>
<td>RWSSJc2</td>
<td>32.82% 40.14% 46.21%</td>
<td>37.26% 45.72% 54.12%</td>
<td>35.86% 46.00% 55.47%</td>
</tr>
<tr>
<td>RWUSJc2</td>
<td>34.14% 40.25% 45.07%</td>
<td>37.15% 44.87% 51.37%</td>
<td>36.45% 45.79% 52.40%</td>
</tr>
<tr>
<td>RWUJC3</td>
<td>32.27% 33.21% 35.45%</td>
<td>34.50% 39.17% 41.69%</td>
<td>36.21% 40.60% 42.82%</td>
</tr>
<tr>
<td>RWSSJc3</td>
<td>35.72% 45.06% 51.29%</td>
<td>36.28% 44.58% 54.13%</td>
<td>35.82% 44.52% 52.73%</td>
</tr>
<tr>
<td>RWUSJc3</td>
<td>35.64% 43.91% 50.16%</td>
<td>35.61% 45.89% 53.98%</td>
<td>36.17% 44.99% 53.67%</td>
</tr>
<tr>
<td>RWUJC4</td>
<td>33.16% 35.88% 33.60%</td>
<td>34.97% 39.10% 41.68%</td>
<td>36.33% 40.62% 43.60%</td>
</tr>
<tr>
<td>RWSSJc4</td>
<td>35.15% 36.85% 30.56%</td>
<td>35.49% 39.74% 33.45%</td>
<td>35.40% 40.85% 36.11%</td>
</tr>
<tr>
<td>RWUSJc4</td>
<td>32.39% 34.58% 33.54%</td>
<td>33.88% 37.52% 34.75%</td>
<td>33.53% 37.42% 33.28%</td>
</tr>
</tbody>
</table>

#### Table 10. Accuracy Performance Comparison (Failure Rate)

<table>
<thead>
<tr>
<th>Methods</th>
<th>G10 Density = 10% G20 G30</th>
<th>G10 Density = 20% G20 G30</th>
<th>G10 Density = 30% G20 G30</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>27.70% 27.98% 28.20%</td>
<td>27.06% 25.20% 23.97%</td>
<td>27.13% 27.02% 25.20%</td>
</tr>
<tr>
<td>RWUJC1</td>
<td>29.51% 29.73% 28.62%</td>
<td>27.74% 26.56% 30.34%</td>
<td>29.51% 29.01% 24.46%</td>
</tr>
<tr>
<td>RWSSJc1</td>
<td>29.05% 28.33% 27.38%</td>
<td>28.02% 32.90% 29.33%</td>
<td>28.36% 28.70% 32.10%</td>
</tr>
<tr>
<td>RWUSJc1</td>
<td>27.75% 28.42% 28.16%</td>
<td>29.63% 31.06% 28.88%</td>
<td>30.58% 28.68% 29.85%</td>
</tr>
<tr>
<td>RWUJC2</td>
<td>29.28% 28.26% 27.48%</td>
<td>31.70% 28.85% 31.35%</td>
<td>31.79% 35.50% 31.93%</td>
</tr>
<tr>
<td>RWSSJc2</td>
<td>35.79% 37.92% 35.28%</td>
<td>35.93% 37.49% 39.71%</td>
<td>36.16% 39.61% 40.75%</td>
</tr>
<tr>
<td>RWUSJc2</td>
<td>34.07% 35.21% 36.72%</td>
<td>36.23% 38.32% 39.28%</td>
<td>36.26% 39.02% 39.41%</td>
</tr>
<tr>
<td>RWUJC3</td>
<td>34.90% 32.74% 34.11%</td>
<td>34.06% 31.80% 33.72%</td>
<td>33.65% 34.58% 32.20%</td>
</tr>
<tr>
<td>RWSSJc3</td>
<td>36.23% 37.90% 38.31%</td>
<td>35.54% 39.19% 41.07%</td>
<td>37.45% 40.74% 39.90%</td>
</tr>
<tr>
<td>RWUSJc3</td>
<td>35.05% 37.01% 39.51%</td>
<td>35.75% 37.86% 42.15%</td>
<td>37.15% 39.83% 38.18%</td>
</tr>
<tr>
<td>RWUJC4</td>
<td>29.58% 28.52% 28.50%</td>
<td>32.22% 31.29% 30.20%</td>
<td>31.49% 29.61% 28.12%</td>
</tr>
<tr>
<td>RWSSJc4</td>
<td>30.67% 13.56% 12.30%</td>
<td>30.42% 11.57% 10.42%</td>
<td>31.00% 12.16% 10.00%</td>
</tr>
<tr>
<td>RWUSJc4</td>
<td>19.53% 13.16% 11.06%</td>
<td>13.51% 11.42% 10.95%</td>
<td>14.19% 10.09% 10.06%</td>
</tr>
</tbody>
</table>
Figure 5(a) and 5(b) report recall results of RT matrix with density training matrix = 10%, while 5(c) and 5(d) employ density training matrix = 20%. The experimental results of Figure 5 show that the recommendation accuracy of RWSSJc3 and RWUSJc3 approaches are significantly enhanced with large given numbers. This observation indicates that by providing more Web service QoS values, the service user can obtain an enhanced recommendation list of unknown web services. We also notice that basic RW is less sensitive to higher given numbers compared with similarity-based approaches. This observation confirms that the proposed similarity-based RW approach performs better with positive user involvement. This idea encourages users to actively contribute in delivering their QoS data.

5.4.3 Training Matrix Density

To study the influence of the training matrix density on recommendation accuracy, we first vary the density from 5% to 50% with a step value of 5%. The remaining settings are: training user number = 100, Top-k = 10 and 15, given number = 10 and 20.

Figure 6 shows the recall results of RT matrix, where Figure 6(a) and 6(b) employ given number of 10, while 6(c) and 6(d) employ given number of 20. The experimental results indicate that, although the recall trend is not as good as the one in the former experiments; yet in general, the accuracy performance is improved when the density increases from the low 5% upward. However, the enhancement acceleration becomes slower with large density values. This observation implies that enhanced recommendation accuracy can be obtained by collecting more QoS data, which also prompts the end user to contribute his QoS recordings for better recommendations. The other interesting observation is that RWSSJc3 and RWUSJc3 approaches mostly surpass the rest approaches especially at low densities (i.e., a case of sparse data), which in turn proves that these approaches can effectively handle sparse data and still give desired accuracy.

![Figure 4. Comparison of Top-K](image)

![Figure 5. Comparison of Given Number](image)

![Figure 6. Comparison of Training Matrix Density](image)
6 THE EFFICIENT RECOMMENDATION MODEL

In real-world recommendation systems, there is usually a huge number of users, compared to a relatively smaller number of items. For example, the number of movies in a network like Netflix is less than 100000, while the number of its subscribers is currently over 60 million. The same realization applies to the case of Web services, where millions of Internet users exist with a limited number of Web services. In this context, finding an applicable (i.e., efficient) recommendation approach to deal with such numbers, while maintaining the quality of the recommendation outcomes, becomes a necessity, from a system designer’s perspective. In the following paragraph, we present a proposed approach to efficiently reduce the original IMQG recommendation model presented above.

In the reduced recommendation model, we attempt to reduce the recommendation domain on an individual basis, before starting the recommendation process. In other words, we first transform the recommendation model from a complete/unified to a reduced/personalized model on which the RW-based algorithm is then applied. Model reduction is accomplished through selecting a limited set of users who are marked as the closest users to current user $u_t$.

Consequently, we need to determine a selection measure by which the selection of the minimized set of users is specified. From a system designer’s point of view, the complete similarity domain (i.e., including all users), is considered a similarity search space in which each couple of users are connected using an edge. The edge is weighted with the pre-computed similarity magnitude between the two users. The role of the selection measure here is to help determine how far a user is within this space to current user $u_t$.

To perform that, we use a Similarity Selection Threshold (SST) by which we exclude less-similar users and only choose the closer ones. We call the resulted set of closely similar users to user $u_t$, Reduced Similarity Set (RSS$_{u_t}$).

Definition 5 (Similarity Selection Threshold): Similarity Selection Threshold is a decimal value between $[0 - 1]$, where a user $u_j$ is considered similar to a given user $u_i$ if their similarity magnitude is equal or greater than SST value, i.e., $u_j \in$ RSS$_{u_i}$, which represents the reduced recommendation model for user $u_i$.

$$\text{RSS}_{u_i} = \{ u_j : S_{u_iu_j} \geq \text{SST} \}$$

where $S_{u_iu_j}$ is the similarity magnitude of $u_i$ with $u_j$.

Since the proposed recommendation model adopts Jaccard coefficient as the similarity measure, the value of SST is the least Jaccard similarity magnitude at which two users are considered closely similar to each other. For example, if $\text{SST} = 0.9$, then all users who have similarity magnitudes between $[0.9 - 1.0]$ with the given user will belong to the reduced set of the user RSS$_{u_i}$.

In other words, when computing recommendations to $u_t$, not all users with their ratings will be involved, rather only the ones who are in his reduced similarity set. As a result, applying the final recommendation approach (i.e., RW algorithm) to the reduced set will consume less computation time. Aiming at faster processing, we also need to ensure that the final recommendation results are of a close quality to the results of a complete-domain/non-reduced recommendation approach. Figure 7 illustrates a sample recommendation model, in which several users are placed within different distances from current user. The distance represents the similarity magnitude, while the circles represent various values of SST. The figure shows how selecting SST value (e.g., SST = 0.9) produces a smaller similarity domain.

In fact, the reduced recommendation approach can be effectively realized as follows. The Reduced Set of Similar users for each user can be stored in his profile, so that every time we need to generate recommendations, the RSS is ready and need not to be created again. This achieves a significant reduction in computation time. However, there must be an update strategy by which this set is recomputed again either periodically or after a certain number of update operations, recorded on the entire set. As the ratio of the number of new items to the number of existing ones is practically low, such an update strategy can be effectively applied. For instance, in E-commerce, users tend to do a lot of search before making a real purchase. Additionally, many Recommender systems currently follow the push approach in which the recommendations will come up automatically once you navigate to an online page. Therefore, the total number of updates to the Recommender knowledge base is significantly less than the number of readings.

In the following sections, we conduct a series of experiments to ensure the effectiveness of the efficient recommendation approach in terms of quality of outcomes and amount of speedup (i.e., computation time reduction). In addition, we also need to investigate to what extent the
reduction of recommendation model affects the ability of RW algorithm to deal with data sparsity issue. In other words, the goal is to have the proposed approach maintain the ability to cope with the common sparsity problem.

7 Evaluation of the Efficient Recommendation Model

In the following experiments, the SST is employed to tune the improved recommendation model. By varying the value of SST, the size of RSS is controlled which in turn affects the recommendation results.

Note that whenever SST is set to 0.0, that means the non-reduced recommendation approach is applied (i.e., all similarity values are included). On the other hand, if SST is set to 1.0, this means that only completely matching users are considered. Specifically, the size of the ratings domain is significantly controlled by the value of SST, which also determines the overall time of recommendation tasks.

7.1 Recommendation Accuracy

In terms of recommendation accuracy, we conduct a series of experiments with various settings to study the impact of Domain Reduction Ratio (DRR), Top-K parameter, and Jaccard coefficient on the reduced model as follows.

7.1.1 Domain Reduction Ratio (DRR)

The purpose of this experiment is to study the effect of applying the efficient recommendation model with various values of SST (i.e., different sizes of Reduced Similarity Set), in terms of recommendation accuracy and Domain Reduction Ratio. Domain Reduction Ratio is the measure that computes how much the domain is reduced compared to the entire domain and is defined as follows:

\[ \text{DRR}_{ui} = \frac{U_N - \text{sizeof}(\text{RSS}_{ui})}{U_N} \]  

where \( U_N \) is the total number of users in the domain.

The settings of the experiment are as follows: SST is varied from 0.99 to 0.00, given number = 20, training user number = 100, training matrix density = 20%. In addition, recommendation approach used is RWUSJc3.

Empirically, the recall measure is more reflective to the recommendation accuracy, compared to precision, which clarifies why we based most of the experiments on this measure. Figure 8 shows the results of the experiment.

Results show that there is no significant impact of DRR values over the recommendation accuracy. Specifically, when DRR is high (i.e., only very similar users to given user are considered), the recall results are still stable. This observation indicates that the reduced recommendation model has a positive performance in producing accurate recommendations, i.e., considering partial domain is enough to preserve the quality of recommendations.

7.1.2 Impact of Top-K

In this experiment, we study the impact of Top-K parameter with various SST values. Consequently, in terms of recommendation accuracy, both recall and precision are measured against a range of Top-K values.

The settings of the experiment are as follows: Top-K is varied from 2 to 20 with a step value of 2, given number = 20, training user number = 100, training matrix density = 20%. The used recommendation approach is RWSSJc3. Figure 9(a) and 9(b) report recall and precision results respectively.

Results indicate that Top-K parameter still has a positive impact on the recommendation accuracy. Similar to prior experiments, no significant effect is on accuracy by reducing the domain. The fact that all competing reduced approaches have close recall results also provides evidence to the effectiveness of our proposed reduced model.

7.1.3 Impact of Jaccard Coefficient

The goal of this experiment is to study the impact of various Jaccard similarity forms over the efficient recommendation model using multiple SST values. The experiment employs recall and precision to test the recommendation accuracy.

The settings of the experiment are as follows: Top-K is set to 15, given number = 20, training user number = 100, training matrix density = 30%. The recommendation approaches are RWUSJc3, RWUSJc2, RWUSJc1 and RWUSJc4. Figure 9(c) and 9(d) report recall and precision results respectively.

Results demonstrate that RWUSJc3 and RWUSJc2 approaches steadily outperform both RWUSJc4 and RWUSJc1. Similar to non-reduced recommendation approach, RWSSJc4 comes in the middle in both sub-experiments. The other observation is that the proposed model maintains consistent recommendation accuracy across the range of SST values, acknowledging the successful employment of the model in generating the recommendations.
7.2 Computation Time

To show the effectiveness of the proposed model in terms of reducing computation time, we conduct a series of experiments with various settings by varying Given Number, Density and Top-K parameters. In all experiments, the recommendation approach used is \( RWSS/j_c \) with the four different types of Jaccard coefficient, within Reduced (R) and Non-reduced or Normal (N) recommendation models. As a result, there are eight different recommendation approaches.

7.2.1 Given Number

This experiment is designed to calculate the time (in seconds) required to generate a list of recommendations to end user while varying the corresponding given number. The settings are: given number is varied from 5 to 40 with a step value of 5, training user number = 100, Top-k = 10, training matrix density = 20%.

Figure 10(a) reports Time results using \( RT \) matrix with all Jaccard based approaches, each of which in normal and reduced versions. The experimental results of Figure 10(a) show that computation time of all reduced versions outperforms their non-reduced counterparts in all cases.

The basic justification is that reduction of the recommendation model from all users to a set of limited, yet highly similar ones, leaves a lower number of ratings, i.e., less computation time to process them.

Furthermore, \( RWSS/j_c R \) and \( RWSS/j_c 2R \) interestingly outperform the remaining approaches. Unlike \( j_c_1, j_c_2 \) has a good reputation in terms of accuracy, the result indicates that their initial Jaccard calculation puts less emphasis on separating ratings according to their quality, which resulted in low similarity values. As a result, the SST filtering process excludes most of the missed similarity values, leaving only a small RSS set which obviously interprets the faster performance.

We also notice that the trend is decreasing with larger number of Given Number, for all competing approaches. This is justified by the fact that the number of test entries will decrease with large given numbers which in turn results in fewer comparisons to produce the final results.
7.2.2 Density

The goal of this experiment is to measure the computation time required to process recommendation tasks with different training matrix density values. The settings are: training matrix density is varied from 0.05 to 0.5 with a step value of 0.05, given number = 20 training user number = 100, Top-k = 10.

Figure 10(b) shows Time results using RT matrix with all Jaccard based approaches, each of which in normal and reduced versions. Results are similar to the previous experiment in terms of superiority of the reduced approaches over their non-reduced counterparts. However, the RWSS$S_{c, R}$, which is the reduced service-service approach using Jaccard 3, outperforms the remaining approaches in time consumption reduction, excluding the exceptional case of RWSS$S_{c, R}$ and RWSS$S_{C, R}$. We also notice that the trend of all is steady within a certain range. This can be justified by the fact that increasing density has no significant effect on the number of users excluded. This observation proofs that the proposed efficient recommendation model performs well enough in low densities. Therefore, it can be considered a successful sparsity-tolerance model.

7.2.3 Top-K

The goal of this experiment is to measure the computation time required to process recommendation tasks with different Top-K values. The settings are: Top-k is varied from 2 to 20 with a step value of 2, given number = 20 training user number = 100, training matrix density = 20%.

Figure 10(c) reports Time results using RT matrix with all Jaccard based approaches, each of which in normal and reduced versions. The experimental results of Figure 10(c) confirm the result of the above two experiments that the reduced approach outperforms its non-reduced counterpart at all times. However, we observe that the distinction is larger now between RWSS$S_{c, R}$ and RWSS$S_{C, R}$ and the rest. We also notice that there is no clear trend of all approaches. This can be interpreted by the fact that in-creasing Top-K has no role in determining the number of users selected in RSS. This observation indicates that using higher Top-K values has no extra cost of time consumption, while producing better recall results. This also adds to the features of the reduced recommendation model.

8 Conclusion

In this paper, we propose an Integrated-Model QoS-based Graph for web service recommendation. First, we construct User-Service Bipartite Graph that contains weighted QoS attributes representing the relations between users and services they have previously invoked. Then, Jaccard similarity coefficient is used to construct similarity Unipartite Graphs among users and services separately. By fusing all the three graphs in one model, we create the Integrated-Model QoS-based Graph, on which a Top-k
random walk approach is applied. Finally, a list of high ranked Web services is recommended to end users. To further improve the model, we also apply a model reduction technique, in which a Similarity Selection Threshold is used to filter out unlikely supportive users. The result is a reduced recommendation model on which the RW algorithm is then applied. To evaluate our framework, comprehensive experiments are conducted on a real-world QoS dataset. Results show that our WS recommendation approach achieves significant improvement in recommendation accuracy, effective tolerance to data sparsity and considerable reduction in computation time.

REFERENCES


AUTHORS

Abdullah Abdullah is a PhD candidate in the School of Computer Science at the University of Guelph. His main research area is in Distributed Systems, and specifically in Service and Agent Computing. His current research focuses on using recommendation techniques and Agent technology to design a web service recommender framework for individual and composite Web Services.

Xining Li is a professor of the School of Computer Science at the University of Guelph, Canada and the director of the IMAGO Lab. His research interests include mobile agent system, logic programming, context-aware computing, virtual machine implementation and mobile commerce systems. Li received a PhD in computer science from the University of Calgary, Canada.