Ranking Web Services using Centralities and Social Indicators

Tilo Zemke¹, José Ignacio Fernández-Villamor² and Carlos Á. Iglesias²
¹Technische Universität Chemnitz, Straße der Nationen 62, 09111 Chemnitz, Germany
²Universidad Politécnica de Madrid, Avenida Complutense 30, 28040 Madrid, Spain

Keywords: Web Service Composition, Ranking.

Abstract: Nowadays, developers of web application mashups face a sheer overwhelming variety and pluralism of web services. Therefore, choosing appropriate web services to achieve specific goals requires a certain amount of knowledge as well as expertise. In order to support users in choosing appropriate web services it is not only important to match their search criteria to a dataset of possible choices but also to rank the results according to their relevance, thus minimizing the time it takes for taking such a choice. Therefore, we investigated six ranking approaches in an empirical manner and compared them to each other. Moreover, we have had a look on how one can combine those ranking algorithms linearly in order to maximize the quality of their outputs.

1 INTRODUCTION

Over the past years, the number of web services that offer an API to access their functionalities has risen rapidly. As of February 2012, ProgrammableWeb.com¹ as one of the most important directories for APIs holds over 5,000 APIs in its database. This sheer overwhelming plurality of web services that are available to the community of mashup creators does not only provide a huge amount of possibilities to mash up the World Wide Web but also requires a certain level of expertise when one wants to create a mashup. Therefore, choosing appropriate web services to accomplish a specific goal can still be a time-consuming task scaring off potential developers that are not that experienced in mashup creation.

In order to overcome this issue, there are different approaches pursued by current research. Typically, a user request gets semantically matched against a web service’s description and a ranking is produced by providing a list of web services descending in similarity scores. Other approaches employ mechanisms of crowd computing, such as tagging web service descriptions, for example.

Focusing on how to bring order in the variety of web services, we investigated six simple ranking approaches, that work independently from a user request, and analysed them by means of the quality of their outputs. Furthermore, we investigated the possibility of linearly combining them to compound ranking functions that provide quality enhancements.

The rest of this paper is structured as follows: First, we give a brief introduction on the the model we employed for our rankings in section 2, followed by the details of the implemented ranking functions as well as our approach to combine them. Afterwards, the methodology used for evaluating and comparing the implemented ranking functions and their respective outputs is described in section 3. Section 4 summarizes the related work and finally, conclusions of our work are drawn in section 5 along with a glimpse on the future work.

2 RANKING MODEL

A metadirectory consisting of web services, mashups as well as widgets served as the starting point for our work. This metadirectory makes use of the Linked Mashups Ontology (LiMOn) (José I. Fernández-Villamor and Tilo Zemke and Carlos Á. Iglesias, 2012), a unified model for those components, which integrates information that are available from current repositories in the web and covers trust, business as well as technical aspects. Formalizing this, our dataset consists of a set of web services $S$ and a set of mashups $M$.

In order to support a developer in choosing the right web services a two-step-methodology was used:
• First, we filtered \( S \) for potential component candidates using her query. This filtering provides a subset of \( S \) called \( S_{\text{query}} \).

• Afterwards, a ranking function \( f \in F \) is applied on \( S_{\text{query}} \) and returns a specific permutation of \( S_{\text{query}} \), i.e. a ranking of the web services \( s_i \in S_{\text{query}} \), by assigning each service \( s_i \) a ranking score \( r_i \).

Focusing on the ranking part, the filtering was done by selecting only web services from the metadirectory whose names, textual descriptions and/or tags contained a specific search term. Hence, for example, the subset \( S_{\text{image}} \) contains all web services that have the term “image” in their respective names, textual descriptions and/or tags.

Our goal was to find indicators for a web service’s relevance and therefore, six different simple ranking functions were chosen and implemented, namely four different types of centrality and two indicators of social activity. Dealing with centralities, we defined an undirected and bipartite graph \( G = (V,E) \) letting the set of vertices \( V = S \cup M \) be the union of the set of web services and the set of mashups in our dataset. The set of edges was defined as \( E = \{ (s,m) \mid \text{Mashup } m \text{ uses service } s \} \) using the property \( \text{uses} \) of LiMOn. Figure 1 illustrates the structure of this graph.

![Figure 1: Illustration of the Mashup API Graph.](image)

2.1 Simple Ranking Functions

In particular, the following ranking functions have been investigated:

• \( C_D \): Degree Centrality, i.e. in our scenario the number of mashups that use a certain web service, which directly reflects its popularity.

• \( C_B \): Betweenness Centrality is a more complex approach that considers the number of shortest paths between two vertices \( v \neq u \) a web service \( s \) lies on. This metric is an important technique in social network analysis and can be determined with the help of Brandes’ algorithm (Brandes, 2001).

• \( C_C \): Closeness Centrality: A vertex \( v \) is ranked higher the shorter the geodesic distances between itself and other vertices are, i.e. the closer it is to other vertices. Closeness centrality is also an important technique in social network analysis and can be determined with the help of Brandes’ algorithm as well – even as a side product of calculating \( C_B \). In order to be working with our graph structure we implemented it with a modification (Opsahl et al., 2010) proposed.

• \( C_E \): Eigenvector Centrality is a very established and successful approach to rank documents in other domains, e.g. PageRank (Page et al., 1998) for web resources. The central idea behind it is that a web service gets ranked higher the more important the mashups are that use it and vice versa.

• \( \text{PUR} \): The score ranging from 0 to 5 each web service has on Programmable Web’s user rating functionality which measures the degree of satisfaction the users had when working with a specific API.

• \( \text{GSO} \): The amount of hits the Google Search Engine returned querying it for the web service’s name and limiting the results to the domain of StackOverflow, a question-and-answer website specialized on programming topics, is an indicator of how widespread a web service is among developers. “Twitter site:stackoverflow.com” could serve as an example for such a search engine query.

2.2 Compound Ranking Functions

In addition to the mentioned simple functions, we investigated on how one could combine them linearly in order to create a new, compound ranking function, which can result in a different permutation of \( S_{\text{query}} \). Such a linear combination \( f \) can be described as in equation 1.

\[
F(S_{\text{query}}) = \sum_{f \in F} \lambda_k f(S_{\text{query}}) \quad (1)
\]

The following example will illustrate the idea behind this: Having three web services in our subset, \( s_A, s_B \) and \( s_C \), as well as two simple ranking functions, \( f_1 \) and \( f_2 \) which produce the following ranking scores \( r_i \):

• Function \( f_1 \) ranks service \( s_A \) as the most relevant one with a score of \( r_A = 10 \), \( s_B \) in second place scoring \( r_B = 5 \) and \( s_C \) in third place with a score of \( r_C = 1 \).

• Function \( f_2 \) places \( s_B \) (\( r_B = 8 \)) first, \( s_C \) (\( r_C = 3 \)) second and \( s_A \) last with a score of \( r_A = 1 \).

\(^2\text{http://www.google.com}\)

\(^3\text{http://www.stackoverflow.com}\)
In order to create a new ranking, we can now combine \( f_1 \) with \( f_2 \) using \( \lambda_1 = \lambda_2 = 0.5 \). The resulting ranking of our linear combination \( \mathcal{F} \) would be the following:

- Service \( s_B \) scores \( 0.5 \times 8 + 0.5 \times 5 = 6.5 \)
- Service \( s_A \) scores \( 0.5 \times 10 + 0.5 \times 1 = 5.5 \)
- Service \( s_C \) scores \( 0.5 \times 1 + 0.5 \times 3 = 2.0 \)

Table 1: Illustration of the example scenario for a linear combination of ranking functions (\( \lambda_1 = \lambda_2 = 0.5 \)).

<table>
<thead>
<tr>
<th>Pos</th>
<th>( f_1 (r_i) )</th>
<th>( f_2 (r_i) )</th>
<th>( \mathcal{F} (r_i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( s_A ) (10)</td>
<td>( s_B ) (8)</td>
<td>( s_B ) (6.5)</td>
</tr>
<tr>
<td>2</td>
<td>( s_B ) (5)</td>
<td>( s_C ) (3)</td>
<td>( s_A ) (5.5)</td>
</tr>
<tr>
<td>3</td>
<td>( s_C ) (1)</td>
<td>( s_A ) (1)</td>
<td>( s_C ) (2)</td>
</tr>
</tbody>
</table>

3 EVALUATION AND RESULTS

The metadirectory contains over 10,000 web services and 7,000 mashups after crawling ProgrammableWeb and Yahoo Pipes\(^4\) in July 2011.

Adapting the methodology of relevance judgements (Küster and König-Ries, 2009), a group of three relevance judges, i.e. experienced mashup developers, was formed. Moreover, three different sub-sets of our dataset’s web services, i.e. \( S_{twitter} \), \( S_{voice} \) and \( S_{image} \) containing 32, 19 and 18 web services respectively, were chosen. First of all, each relevance judge had to individually rate each web service according to three different criteria, functional scope, technical variety and support as well as trust in the service and its provider. In a second step, the relevance judges met and conflicts, that occurred when two or more judges did not give the same rating on a certain criterium for a specific web service, were discussed until all judges agreed on a uniform rating.

Using this uniform rating to produce gain quantifications \( G_i \), which reflect the relevance, for each web service \( s_i \), the Normalized Discounted Cumulated Gain (\( nDCG_i \)) (Järvelin and Kekäläinen, 2002) metric has been applied to each simple ranking function. The \( nDCG_i \) metric is based on \( DCG_i \) which is defined as follows in our scenario:

\[
DCG_i = \begin{cases} 
G_i, & i = 1 \\
DCG_{i-1} + G_i / \log_2(i), & \text{otherwise (2)}
\end{cases}
\]

The higher the position of web services with high gain quantifications are in a specific ranking, the better the evaluational score of the ranking itself. This leads to a very intuitive sight on the quality of the rankings produced by the simple ranking functions. We chose sharp gain quantifications, i.e. powers of 2, as well as a discounting factor of 2 and we only compared the results up to the 15th position in the rankings (\( nDCG_{15} \)) thus modelling a rather impatient developer that needs quality results in the beginning of his results list.

Figures 2, 3 and 4 show the results of the evaluation done. As can be seen, the ranking functions produce results of considerably similar quality except the ProgrammableWeb user rating \( PUR \). An explanation for \( PUR \)’s lack of quality may be the lack of votes and therefore missing reliability. Moreover, the reason for the similarity between the centrality measures is their strongly-related nature and the structure of our dataset’s graphical representation. For example, the more mashups use a certain web service (\( C_D \)) the higher is the probability of being part of a shortest path in \( G (C_B) \) and the higher the number of mashups or APIs close to it (\( C_C \)).

\(^4\)http://pipes.yahoo.com
Although runtime performance has not yet been taken into consideration, our experiments showed that degree centrality as well as eigenvector centrality deliver the best cost-benefit ratios among the analysed ranking approaches. While betweenness and closeness centrality suffer from their algorithmic complexity ($O(\text{SM})$), the traffic caused by GSO does not imply a practical use.

In addition to that we analysed nearly 325,000 possible linear combinations for each subset of web services that was evaluated and checked whether or not $n\text{DCG}_{10}$ could be improved. The results show that there are slight improvements possible in our scenario with the most remarkable one found in $S_{\text{image}}$ achieving an $n\text{DCG}_{10}$ of 0.9715 for a combination of $C_C$, $GSO$ and $\text{PUR}$, i.e. $\frac{1}{3}C_C + \frac{2}{5}GSO + \frac{2}{15}\text{PUR}$, over 0.9307, the best score of a simple ranking function ($C_C$) in this specific subset. Table 2 shows the scores of the most successful linear combinations ($n\text{DCG}_{10}$) compared to the most successful elementary ranking functions for each evaluated set.

As can be seen, the improvements achieved by linearly combining ranking functions, especially for $S_{\text{twitter}}$ and $S_{\text{voice}}$, are not very high. This is a result of the likewise nature of our elementary ranking functions and therefore the similarity of the rankings they produce.

### 5 CONCLUSIONS AND FUTURE WORK

Throughout this paper we showed that the presented ranking algorithms can produce quality rankings. Moreover, we showed that ranking functions can be linearly combined in order to improve those rankings. Due to the similarity between the analysed rankings, those improvements were mostly rather minimal. Therefore, other ranking approaches, such as, for example, semantic similarity scores for the web services’ descriptions to the user’s search query or QoS of a web service, should be taken into account as well. During this work a query interface, called rOMKing for end-users has been implemented, where the presented concepts are provided.

Future work will also involve further analysing the performance of the presented ranking functions as well as the process of efficiently optimizing the rankings with the help of linear combinations. Enhancing the capabilities of the minimalistic filtering process is planned, too.

---

5http://fusion.cs.uni-jena.de/lehrstuhl/jgdeval/
ACKNOWLEDGEMENTS

This research project was funded by the European Commission under the R&D project OMELETTE (FP7-ICT-2009-5).

REFERENCES


