Search Result Ranking for a Reputation Analysis Prototype

Author:

Andrea Liechti

Zelgweg 40, 3150 Schwarzenburg

andrea.liechti@unifr.ch — Student-Number: 08-213-340

Supervision: Assistance:

Prof. Dr. Andreas Meier Edy Portmann

andreas.meier@unifr.ch edy.portmann@unifr.ch

February 26th, 2012

Information Systems Research Group

Département d’Informatique - Departement für Informatik • Université de Fribourg - Universität Freiburg • Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland
Abstract

As the Web grows and evolves, it becomes more and more a Social Semantic Web including both the social and the semantic aspects. In a first part, this bachelor thesis goes deeper into the history of the Web and the synthesis of a Social Web and a Semantic Web.

Information storage and retrieval is a major part of the Social Semantic Web and what makes it so powerful. In the thesis, two comparisons are made: Which search ranking algorithm, PageRank or HITS, and which query language, SPARQL or RDQL, are best used in the practical part of the project? Both, search ranking algorithms and query languages, as well as the major examples of these are explained beforehand.

After finding the best options for the question above, a case example shows how they could be used.

The practical part of the thesis explains the project of the University of Fribourg and the National University of Singapore collaboration called YouReputation and the FORA framework. Following this is an explanation of the code in the appendix. The code is a small part of the YouReputation website (www.youreputation.org) and handles information retrieval using SPARQL.

Keywords:

Social Semantic Web, Search Result Ranking, Query Languages, Information Retrieval, Online Reputation Analysis
Contents

1 Introduction ................................................. 5
   1.1 Problem Description .......................................... 5
   1.2 Objectives .................................................. 6
   1.3 Methodology .................................................. 7

2 Theoretical Section ......................................... 8
   2.1 Social Semantic Web ........................................... 8
      2.1.1 History ...................................................... 8
      2.1.2 The Social Web ............................................. 9
      2.1.3 The Semantic Web .......................................... 10
      2.1.4 The Social Semantic Web .................................. 12
   2.2 Search Result Ranking ....................................... 13
      2.2.1 History and early Algorithms ................................ 13
      2.2.2 Link-based Ranking Algorithms ............................. 15
      2.2.3 PageRank .................................................... 16
      2.2.4 HITS ......................................................... 17
   2.3 Search Ranking Algorithms Comparison ...................... 18
      2.3.1 Comparison Criteria & Explanation of Choice ............. 18
      2.3.2 Comparison: Using the Criteria against the Algorithms .... 19
      2.3.3 Comparison .................................................. 19
      2.3.4 Choice & Conclusion ....................................... 20
   2.4 Query Languages ............................................. 21
      2.4.1 SPARQL ..................................................... 21
      2.4.2 RDQL ....................................................... 23
   2.5 Query Language Comparison ................................ 24
      2.5.1 Criteria ..................................................... 25
      2.5.2 Comparison .................................................. 25
      2.5.3 Choice & Conclusion ....................................... 26

3 Case Example .................................................. 26
   3.1 Online Reputation Analysis ................................... 26
      3.1.1 Reputation .................................................. 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.2</td>
<td>Reputation Analysis</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Introduction to the Prototype</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>Functionality of the Prototype</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>Practical Section</td>
<td>30</td>
</tr>
<tr>
<td>4.1</td>
<td>Search Result Ranking for the Social Semantic Web</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Annotations to the Code</td>
<td>30</td>
</tr>
<tr>
<td>4.2.1</td>
<td>User Input, Database Connection and Query</td>
<td>30</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Preparing the Result for Computations</td>
<td>31</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Find Close Tags</td>
<td>33</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Center Distances</td>
<td>34</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Handling and Returning the Result</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion &amp; Outlook</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Literature</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Appendix</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>The Code</td>
<td>41</td>
</tr>
</tbody>
</table>
List of Figures

1 Subject, Predicate, Object ........................................... 11
2 Social Semantic Web ..................................................... 12
3 SPARQL Query ............................................................. 23
4 RDQL Query Structure .................................................... 24
5 The website YouReputation .............................................. 29
6 SPARQL Query to get data from AllegroGraph ...................... 31
7 JSON Return ............................................................... 35

List of Tables

1 Comparison Criteria for Ranking Algorithms ....................... 20
2 Query Languages and list of the abbreviations ................. 22
3 Comparison Criteria for Query Languages ....................... 26
1 Introduction

This document is the theoretical part of a bachelor thesis, which also includes an implementation described later. It is divided into two parts, a theoretical bedding and a practical prototype. In the first section, we will have a look at the Social Semantic Web, search result ranking and algorithms, as well as query languages. The prototype described in the practical section includes and works with these theoretical aspects, it is part of a collaboration of the University of Fribourg and the National University of Singapore (NUS). The purpose of this first chapter, is to describe the problem, objectives and methodology of the bachelor thesis.

1.1 Problem Description

As the World Wide Web grew larger and larger over the last few decades, finding information, and especially finding relevant and useful information has become more and more important. The Web in its decentralized structure offers an unimaginable amount of data. Information Retrieval (IR) methods help representing, storing, organizing and accessing the data and provide users with the information they seek. In the early days, the Web was indexed in a similar way as library books were indexed. In the current development towards a Social Semantic Web, IR includes modeling, Web search, text classification, systems architecture, user interfaces, data visualization, filtering and languages (Ribeiro-Neto & Baeza-Yates, 2010, p. 1).

In a computer-centered point of view, two of the main parts of IR are the ranking and the query languages.

Ranking algorithms calculate the importance of each search result to a given query. If one result is rated more important and relevant to a given query, than another, it is displayed higher up in the results page. There are many different ranking algorithms, some are quite easy to understand and others are very complicated mathematical models. Part of this thesis is it, to find the ranking algorithm that best fits the concept and need of the YouReputation prototype.

Query languages are the languages to find information in a database according to user needs. A query language is developed for a certain kind of stored data, it matters, what form and language the data has in order for a query language to be working. If a German speaking person wants to ask for directions from an English
speaking person, there will not be an answer if he asks in German.

As a hands-on approach to this Bachelor thesis, there was an implementation of a ranking algorithm for a meta search engine planned. However during the process of work, several problems occurred and the finished part of the prototype now retrieves search terms similar to a user query and searches them on delicious (www.delicious.com). The ranking in this case is done by said search engine. Problems here were to get fuzzy results from a non-fuzzy database. The IR in the prototype uses the query language most fitting to YouReputation. The next subsection will explain the objectives and methods.

1.2 Objectives

In order to get to the more practical approaches it is fundamental to explain what the Social Semantic Web is and to talk about the theoretical aspects of search result ranking and query languages. Even though the prototype does not handle ranking, it is still a central question of this thesis to find an algorithm for YouReputation. In order to find the ranking algorithm and query language that would best be used, the following questions will be answered:

1. The Web of today has merged from two previous forms and two different approaches: the Semantic Web and the Social Web. Question number one in this Bachelor thesis will answer how the World Wide Web of today evolved to be a Social Semantic Web. It will give a short insight to the history of the Web and an outlook of what might come in the future.

2. Secondly, two sub-questions will be analyzed and answered:

   a. What are the most popular and important search ranking algorithms today and how do they work? In order to choose an algorithm that fits the purpose of the project, it is essential to know the existing algorithms and their functionalities. What aspects are to consider when judging and choosing a search ranking algorithm? There are several aspects (for example: speed, complexity, usefulness of results, etc.) to consider when choosing an algorithm. In order to find the most suited algorithm for a certain situation, one has to be familiar with these aspects and their
importance. Considering these aspects: which search ranking algorithm is best used for a meta search engine like YouReputation?

b. What are Social Semantic Web query languages? To retrieve data and information from a database, a query language is needed. There are several different Semantic Web query languages like SPARQL, RDQL, SQWRL and so on. Which two of these languages should be compared? Of the compared ones, which language can be used and is most suited for the project?

3. These questions are answered theoretically and underlined with the example of YouReputation.

4. How can we extract information from a database and prepare it for YouReputation? This is the main component of the practical part of the thesis and follows the question 2b. It is a documentation of the prototype to this Bachelor thesis.

Following, the methodology for these questions is explained.

1.3 Methodology

Using literary research and analysis for research question number one and two makes most sense. Over the last few decades, a lot of books and documents treating IR, ranking and query languages, have been written and published. Question number three will be answered in a case example. As for the fourth question, the goal was initially to be programming a search ranking algorithm for the Fuzzy Online Reputation Analysis (FORA) framework. This would have, of course, been the algorithm found to be most useful. Because of certain changes in the project, the search result ranking algorithm is not yet implemented, but the query language is used and the results are processed.

The following section will explain the theoretical aspects of the Social Semantic Web as well as search result ranking, only after that, the prototype will be explained in more detail.
2 Theoretical Section

The theoretical section will show how the Web evolved over the last few decades. IR, namely search ranking algorithms and query languages will be treated more deeply and then used in a case example.

2.1 Social Semantic Web

After a part on the history of the Web, the Semantic Web and the Social Web and their combination will be explained as a basis for the further parts of the thesis.

2.1.1 History

Hypertext theory was first introduced by Douglas Engelbart in 1968 and it changed the world. This was the beginning of the change from the industrialized society to the information society (Hitzler, Krötzsch, Rudolph & Sure, 2008, p. 9). Allowing the user to jump from one electronic document to another, hypertext revolutionized the way information was stored and accessed. In the early seventies, e-mail was introduced, around the same time, TCP/IP protocols were invented and operating system integration followed soon after with UNIX. More Internet services came soon after. One of these services is the USENET, a predecessor to today’s news FORA. Because the Internet’s documentation was freely available, it grew fast, but was only used by researchers and scientist in general (Wöhr, 2004, p. 3).

Tim Berners-Lee who worked at the CERN in Geneva saw in Hypertext a solution to his problem. At that time, researchers who wanted to share documentations with others had to reformat the documents for an internal publishing system. Berners-Lee was the solution in a decentralized storing of the contributions, that’s why he wrote the HTTP protocol and defined HTML as well as introduced the first browser and Web server in 1990. The browser was called “World Wide Web”. This Web grew incredibly large in a very short amount of time and it changed everything about our today’s lives (Ribeiro-Neto & Baeza-Yates, 2010, p. 9).

Because of the amount of webpages, people need to be able to find something they’re looking for. That’s why there are search engines. Search engines like Google (www.google.com) or Bing (www.bing.com) help users find relevant information to their query and need. Without them, the Web would certainly not be this popular, to
find an information on the Web, a person would have to know the exact URL of the webpage and would also have to know if the webpage contains required information. It would be like finding a book in a library without any form of help or indexes. Indexing, retrieving and ranking are the main factors to make the Web so popular.

But what exactly is the Social Semantic Web? The following section will explain that.

2.1.2 The Social Web

The “Social Web” is an evolved form of the Web mentioned above. It includes the human as a communicative and interacting being. The technologies of the World Wide Web stay the same, but the Web 2.0 can be looked at as a second generation of Web. The “2.0” does not stand for an update of the previous Web but rather for new structures and abstractions (Breslin, Passant, Decker, 2009, p. 11). The Web is now used to share and communicate, to interact with people all over the World Wide Web. Social applications include wikis, blogs and social networks where knowledge is gathered. Prosumers, a combination of the word producer and consumer can interact and collaborate freely online (Portmann, Nguyen, Sepulveda & Cheok, 2012). Around 2002, social networking sites such as Friendster (early social networking service (SNS)), LindedIn (professional relationships), MySpace (music and youth oriented) and of course the world’s most popular SNS, Facebook, became more and more popular. Knowledge-sharing was now easy, YouTube (video-sharing), Flickr (photo-sharing) and Last.fm (music community) are only a few examples of sites that help people share content over the Web. Users in the Social Web have profiles, friend listings, they can comment and share as well as play games or meet new people. Now, we will have a look at another example: Wikipedia. This online encyclopedia gathers the knowledge of all people who are willing to share. That way, it is creating collective intelligence (Breslin, Passant, Decke, 2009, p. 11f). Collective intelligence is the gathered knowledge of many people together. Knowledge in the Social Web is often not machine-understandable. One aspect of the current social websites is, that they are isolated from each other. If someone is trying to find an answer to a question, he probably has to look for it on many different websites, wikis and forums. Interlinking data is difficult. The cause for this lack of interoperability between social websites is, that there are no common standards for knowledge and
information exchange available. Really Simple Syndication (RSS), which publishes recent changes of a website, is a first step towards this interoperability, but it has limitations. Friend of a Friend (FOAF) and Semantically Interlinked Online Communities (SIOC) are two Semantic Web vocabularies that help express personal profile and social networking information (i.e. FOAF) and interlink communities and distributed conversations (i.e. SIOC)(Breslin, Passant, Decker, 2009, p.13f).

Next, the Semantic Web is explained in order to go further in the history of the Web to become a Social Semantic Web.

2.1.3 The Semantic Web

The Semantic Web is not a separate Web but an extension of the current one, in which information is given well defined meaning, better enabling computers and people to work in cooperation. (Berners-Lee, Hendler & Lassila, 2001). The “Semantic Web” gives ‘sense’ to the information stored in it. Computers should be able to see a difference between a Mustang, the horse, and a Mustang, the car. This is done with the help of metadata. Metadata contains information about the information/file. This will also influence the way retrieved information is ranked. While in the World Wide Web, ranking only looks at the links pointing to and away from a webpage, in the Semantic Web, it can also look at the semantics, the metadata of an information source. “The Semantic Web is a useful platform for linking and for performing operations on diverse person- and object-related data gathered from heterogeneous social websites (in what is termed ‘Web 2.0’)” (Breslin, Passant, Decker, 2009, p. 1). Now with these explanations at hand, we will move on to the combination of these three ‘Webs’: “The Social Semantic Web”. Over the past few years, the focus lay more on the Semantic Web. Research and implementation tended towards making data machine-readable. At the same time, personal information and human relationships, the Social Web, was neglected. The Social Semantic Web combines the two and tries to define social information and relations to be machine-understandable (Mäkeläinen, 2005, p. 1).

**RDF** The Resource Description Framework (RDF) is a language that can represent information, particularly metadata like author or title, about resources in the Web. RDF is intended for the Semantic Web, it has to be machine readable and
understandable. RDF uses Uniform Resource Identifiers (URIs) to identify things. Like this, RDF can represent resources, properties and values as a graph. Individuals, kinds of things, properties of those things and values of those properties are identified by URIs. With RDF/XML the graphs mentioned before can be exchanged (Manola, Miller). Figure 1 shows how RDF data is modeled into triples of subject, predicate and object (Feigenbaum).

![Figure 1: Subject, Predicate, Object](image)

**RDFS** or RDF Schema is a set of classes using the above RDF. It provides basic elements for describing ontologies, gives a structure to RDF and can be used as a RDF vocabulary (Brickley, Guha).

**OWL** In order to map and organize the Web more precisely, the human-readable data has to be machine-readable. Web Ontology Language (OWL) can describe classes and relations between them in Web documents, it defines and instantiates Web ontologies. An ontology is something that describes the entities and their relations in the world (Smith, Welty, McGuinness).

**SPARQL** (Short for “SPARQL Protocol and RDF Query Language”), this is a language to query the triplets in an RDF store. For those familiar with SQL, SPARQL can be compared to the query language SQL as it also uses SELECT - WHERE statements, it is the SQL of the Semantic Web.

After the part about the Social Web and the part about the Semantic Web, the Social Semantic Web is explained in the next section.
2.1.4 The Social Semantic Web

Collective Intelligence includes both aspects and there lays the need for not only a Social Web or a Semantic Web, but a Social Semantic Web. Or, as another quote by Tim Berners-Lee states: “The Web isn’t about what you can do with computers. It’s people and, yes, they are connected by computers. But computer science, as the study of what happens in a computer, doesn’t tell you about what happens on the Web” (Gruber, 2006, p. 6). According to Tom Gruber, an American computer scientist, inventor, and entrepreneur with a focus on systems for knowledge sharing and collective intelligence, collective knowledge is the capacity to provide useful information based on human contributions and it gets better as more people participate. This produces a mix of both structured, machine-readable data and unstructured data from human input (Gruber, 2006, p. 9). It is not possible to create useful Semantic Web applications without the data to power them and the other way around: it’s not possible to produce semantically-rich data without the interesting applications themselves (Breslin, Passant, Decker, 2009, p. 15). Figure 2 shows how the Social Semantic Web is a combination of the Social Web and the Semantic Web.

One idea of a combination of the Social Web and the Semantic Web is introduced
in (Breslin, Passant, Decker, 2009, p.16f), it’s called Social Semantic Information Spaces where data is interconnected and relevant information comes from social spaces. One can gather all his or her data and information from various profiles on different social websites as well as get this for friends’ profiles. The Web could be used as a clipboard to get relevant information from different websites. For example if one searches for a location, a wiki page is displayed but also a map and a list of events in that location as well as the people who are there. One would not have to go over the same information multiple times on multiple websites.

Of course this new way to look at and use the Web brings us to a different approach of searching the Web and ranking these search results. The next section talks about search result ranking history, algorithms and gives an outlook into the future.

2.2 Search Result Ranking

Ranking is said to be one of the most important function of a search engine (Ribeiro-Neto & Baeza-Yates, 2010, p. 468). What use are hundreds of thousands of results if the user cannot find what he is looking for? Two major tasks of a good ranking is to figure out what exactly the user wants and what a good quality search result is. Also an important part in today’s world is the avoidance of Web spam and other malicious content. First, we’ll have a look at how it all started.

2.2.1 History and early Algorithms

The concepts and ideas behind search result ranking dates back to the 1940s. Nobel Prize winner Wassily W. Leontief developed the input-output model in 1941. Leontief divides a country’s economy into sectors, each of these sectors supplies others and receives from others, but not in equal measure. To find out the value of each sector, Leontief developed an iterative method. Each sector’s value is based on the importance of the sectors supplying it (Franceschet, 2010, p. 12).

In the field of sociometry, the quantitative study of social relationships, John Seeley used the circular argument of PageRank already in 1949. According to Seeley, a child chooses other children in a social group, and he defines the popularity of a child as follows: a child’s popularity depends on the popularity of the children
choosing it, and the popularity of the choosers is calculated in the same way. It is an “indeinitely repeated reflection”. In 1965, Charles Hubbell published a similar technique where the importance of individuals is based on the importance of people endorsing them (Franceschet, 2010, p. 11).

(Franceschet, 2010) also mentions Bibliometrics as an important background of search result ranking. Gabriel Pinski and Francis Narin developed a journal ranking method in 1976. “A journal is inuenial if it is cited by other inuenial journals” (Franceschet, 2010).

A classic scheme for term weighing in IR is TF-IDF. TF stands for Term frequency, the more often a term appears, the more important it is. IDF stands for inverse document frequency and measures the general importance of the term by dividing the total number of documents by the number of documents containing the term and then taking the logarithm. TF-IDF assigns weight to a term \( t \) in a document \( d \), see equation (2.1). When HTML came up and ranking webpages became a very important part of its usability, several authors proposed link-based ranking using incoming links. Soon it was clear that this was not enough to be reliable. As an addition to TF-IDF, Yuwono and Lee proposed in 1996 three algorithms: Boolean spread, vector spread and most-cited.

\[
    tf - idf_{t,d} = tf_{t,d} \times idf_t
\]  

(2.1)

**Boolean Spread** A ranking algorithm from the Boolean space model extended to include pages pointed by a page in the answer or pages that point to a page in the answer. A retrieval model specifies the details of document representation, query representation and retrieval functions. It determines a notion of relevance. In the boolean model, the document is represented by a set of keywords. The keywords form a query by using AND, OR and NOT, it does not allow partial matches (Inkpen).

**Vector Spread** Similar to the Boolean spread algorithm, but based on the vector space model. The basic idea behind the vector space model is that documents and queries are vectors in a high-dimensional space. The vector can either be binary, so if a term is there, the matrix contains a 1, if not it has a 0. If it is weighed, the
number in the matrix is the relative “importance” of the term in the document. So, for a given query, the query is converted into a vector, now, all documents containing each term are also converted to a vector. After that, the two vectors are matched and sorted by similarity (Lavrenko).

**Most-Cited** This algorithm is a bit different to the previous two. Each page is assigned a relevance score. This score is calculated as follows: it is the sum of the number of words from the query that are in one of the pages citing the page or having a link referring to the page (Yuwono, Lee, p. 5).

A comparison of the three leaves the vector spread model to have a better recall-precision curve (Ribeiro-Neto & Baeza-Yates, 2010, p. 470).

One more early search ranking algorithm is called WebQuery. It ranks a set of webpages based on how connected each webpage is. It can also find webpages that are highly connected to the original set.

In the following part of the chapter we will have a look at two current algorithms. One of them is the famous PageRank and the other is HITS. HITS is a little older than PageRank. They are both link-based.

### 2.2.2 Link-based Ranking Algorithms

Every Web search engine has a ranking of some sort. It makes sense to find out what search results are more important and meaningful to the user than others. Ranking algorithms contain two main factors: on-page factors and off-page factors. The author of a webpage can influence on-page factors, as they contain an analysis of the page content or text. Off-page analysis (graph analysis) examines the links to the webpage. Search Engine Optimization (SEO) is very important in order to get more visitors on a page.

Two ranking algorithms, the PageRank and HITS will be compared as part of this thesis. The following sections introduce the two ranking algorithms and explain how they work.

**PageRank** Every book and other resources dealing with Web ranking highlight Googles link-based PageRank, named after Larry Page. Google, being the most used search engine on the planet, obviously has to have an elaborate ranking algorithm.
Each webpage has a PageRank score, which is updated every time Google crawls the Web. This crawling process is done in such a way that from a page, a hyperlink is randomly selected and followed, except if its pointing back to the previous page. When landing on a dead end (page without links), the crawler randomly selects any other webpage (Ribeiro-Neto & Baeza-Yates, 2010, p. 472).

**HITS** Hyperlink-Induced Topic Search (HITS) or Hubs and Authorities is also link based. Jon Kleinberg, an American computer scientist and Professor of Computer Science at Cornell University, developed it. This happened a bit earlier than PageRanks development. While PageRank is designed to operate over the entire World Wide Web, Kleinberg saw HITS as an algorithm for smaller Web graphs. HITS, as already mentioned, is also called Hubs and Authorities because the algorithm differs between two sorts of webpages. Authorities are pages containing information on a certain topic, while hubs are pages that mainly contain links to authorities. Good hubs point to many authorities as good authorities are pointed to by many hubs. Of course some pages are both hubs and authorities (Ribeiro-Neto & Baeza-Yates, 2010, p. 470f).

**Why These Two?** (Ribeiro-Neto & Baeza-Yates, 2010) can be considered a bible of Web IR. Of course, it also treats the subject of ranking algorithms, especially Hits and PageRank as these two are well established and have been developed individually. PageRank because of its fame and HITS because of its different approach, both link-based algorithms, are interesting to compare.

### 2.2.3 PageRank

As already explained, PageRank simulates a user who either clicks on a link on the webpage he currently is on, or he goes to a different website accessing it through its URL. The probability that the user will follow a link is $\delta$. Therefore the probability of switching to another webpage randomly is $\delta^{-1}$. For $\delta$ a reasonable value would be between 0.15 and 0.9. The value of PageRank is marked $r(\alpha)$ and signifies the probability with which the page is visited by the user. The page is $\alpha$. Equation
(2.2) shows the formula to calculate the PageRank value of a page.

\[ r(\alpha) = \frac{\delta}{T} + (1 - \delta) \times \sum_{i=1}^{n} \frac{r(p_i)}{L(p_i)} \]  

(2.2)

Where T is the total amount of pages on the graph and L(p) is the number of outgoing links on the page. Page \( \alpha \) is pointed to by page \( p_1 \) to \( p_n \). For a simplified explanation, we assume we only have four webpages A, B, C and D.

- Each webpage has an initial PageRank of a certain value. Here we will use 0.25 (PageRank of 1 evenly shared between the four).
- If B, C and D point to A, they would give their 0.25 to A.
- If B points to A and C, and D points to A, B, C: PageRank of A is the PageRank of B divided by two (because B gives half of its PageRank to C) plus PageRank of C plus a third of PageRank of D. That’s why we divide by the outgoing links in the formula above.
- The ranking is normalized by the number of links in the page.

There are different approaches to handling dead ends, that means pages without outgoing links. One of them is to set \( q=1 \) or another one is to remove them entirely and compute their PageRank in the end.

### 2.2.4 HITS

Hypertext Induced Topic Search (HITS) or also called Hubs and Authorities approaches link based ranking a bit differently. Considering a set of pages \( S \) that point to or are pointed to. Authorities have many pages pointing to them, Hubs point to pages. For every page a hub value and an authority value can be calculated, see equation (2.3).

\[ H(p) = \sum_{u \in S|p \rightarrow u} A(u), \quad A(p) = \sum_{v \in S|v \rightarrow p} H(v) \]  

(2.3)

We only compute these values for the set \( S \) of pages retrieved in a first step.

- On these pages, in a first step, each page has a hub score and an authority score of 1.
• Then, as long as necessary
  – update the authority to be the sum of the hub scores of each page that
    points to it
  – and update the hub score to be equal to the sum of authority scores of
    each page that it points to.

• After that, the values are normalized. That means that each hub/authority
  score is divided by the sum of the squares of all hub scores.

Among the problems that can occur are, that if we want to calculate large numbers
of pages or even the whole Web, a maximal number of pages pointing to one page
must be defined. Another problem is the one of topic diffusion, where some result
pages are not related to the query topic, here the score must be associated to the
page content.

The following sections of this thesis will lay out a comparison of the two ranking
algorithms PageRank and HITS.

2.3 Search Ranking Algorithms Comparison

Now follows a comparison of the two Web ranking algorithms in order to find the one
that’s more applicable for the project YouReputation. First, we will have a closer
look at the two algorithms PageRank and HITS.

2.3.1 Comparison Criteria & Explanation of Choice

To find out which algorithm is better for the FORA-framework and YouReputation
it is necessary to use search criteria on them.

Complexity  The complexity of an algorithm contains two dimensions: time com-
plexity and space complexity. Time complexity looks at the program as a function
of the size of the input. Space complexity is about the amount of computer memory
required during program execution, this is also a function of the input size. To
compare time complexity we use the big o notation. After Cobham a computational
problem can be solved usefully if it can be computed in polynomial time (Kolly,
2011, p. 36). Complexity as a criterion for the comparison of algorithms is useful
because it can be mathematically calculated and tells us in a relative manner how much time is needed to use the algorithm on a certain data set.

**Input Parameters**  The input parameters are what we need from the webpages in order to calculate the ranking. Some algorithms just need back-links, other need both back- and forward links while HITS in our example works with back-links, forward links and even the content of the page. The reason they are used as a criterion is because especially using page contents can cause problems. But, on the other hand, maybe only using back-links does not offer such a thorough analysis of the ranking value of a page.

**Difficulties**  Certain difficulties that have already been mentioned above have to be considered as well. Some are easier to deal with than others. When implementing PageRank, one has to avoid dead ends, that means, web pages with no links pointing to another page. A solution to this can be that a random page is chosen if the algorithm reaches a dead end. Using HITS, one might encounter unrelated documents in the ranked results. The iterative loop of the two algorithms itself might not be difficult to implement, but how to find dead ends or unrelated documents and how to deal with them is not so easy.

**Data Set**  An algorithm works with a certain amount of webpages and ranks them. The amount of webpages can differ in different algorithms. This criterion is used because we do not need to rank large sets of webpages on YouReputation.

**2.3.2 Comparison: Using the Criteria against the Algorithms**

Table 1 shows the criteria and how they relate to the two algorithms. PageRank and HITS were compared looking at their complexity, input parameters, problems and data set as described above.

**2.3.3 Comparison**

PageRank completes the algorithm in $O(\log N)$ time for $N$ pages, while HITS is a bit faster and does the same in under $O(\log N)$ time. PageRank uses back-links to calculate the ranking value of page while HITS uses back-links in authorities, forward
 links in hubs and also the page content. This shows us that PageRank might be easier to implement and understand, but HITS could give us more thorough results by including the more human page content. Taking about the page content being part of HITS brings us right up to the problems. In PageRank, the problem of dealing with dead end pages can relatively easily be solved by assigning these pages a certain value and come back to them later. HITS on the other hand sometimes calculates the values and lists results that are not part of the current topic. It is quite difficult to bind the hubs and authorities score to the page content. While PageRank works on the whole set of pages given, HITS choses a smaller subset to run the algorithm on. On one hand, this makes HITS a bit faster, on the other, some important pages might be lost and not be part of the ranking here.

2.3.4 Choice & Conclusion

PageRank completes the algorithm in O(log N) time for N pages, while HITS is a bit faster and does the same in under O(log N) time (Jain, Purohit, 2011, p. 25). PageRank uses back-links to calculate the ranking value of page while HITS uses back-links in authorities, forward links in hubs and also the page content. This shows us that PageRank might be easier to implement and understand, but HITS could give us more thorough results by including the more human page content. Taking about the page content being part of HITS brings us right up to the problems. In PageRank, the problem of dealing with dead end pages can relatively easily be solved by assigning these pages a certain value and come back to them later. HITS

<table>
<thead>
<tr>
<th>Complexity</th>
<th>PageRank</th>
<th>HITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>O(log N)</td>
<td>&lt;O(log N)</td>
</tr>
<tr>
<td>Input Parameters</td>
<td>Backlinks</td>
<td>Backlinks, Forward Links, Page Content</td>
</tr>
<tr>
<td>Problems</td>
<td>Dealing with dead ends</td>
<td>Topic-unrelated documents could show up as well</td>
</tr>
<tr>
<td>Data Set</td>
<td>The entire set of pages</td>
<td>A pre-chosen subset of pages on a certain topic</td>
</tr>
</tbody>
</table>

Table 1: Comparison Criteria for Ranking Algorithms
on the other hand sometimes calculates the values and lists results that are not part of the current topic. It is quite difficult to bind the hubs and authorities score to the page content. While PageRank works on the whole set of pages given, HITS chooses a smaller subset to run the algorithm on. On one hand, this makes HITS a bit faster, on the other, some important pages might be lost and not be part of the ranking here.

A second comparison will now compare two major query languages of the Social Semantic Web, SPARQL and RDQL.

2.4 Query Languages

“A query is the formulation of a user need” (Ribeiro-Neto & Baeza-Yates, 2010, p. 256). A query language is used to query data in a database. These languages make it easier for the user to find the data needed, but on the other hand are not that simple as they have strong syntactic rules. There are multiple database systems and also multiple query languages. Of these query languages, there are those for IR with ranking in mind, and others for data retrieval where the results are not intended to be ranked (Ribeiro-Neto & Baeza-Yates, 2010, p. 255).

As in (Portmann, Nguyen, Sepulveda & Cheok, 2012, p. 15), established query languages are the ones listed in Table 2.

For this comparison, it makes sense to compare SPARQL and RDQL because they can both be used on RDF data. The data stored in AllegroGraph’s RDFStore that is used for YouReputation is in RDF triplets.

2.4.1 SPARQL

SPARQL has already been introduced briefly in the beginning of this thesis. If data is stored in RDF(S) or OWL, one needs to be able to query it and work with it. SPARQL is a query language for this purpose and also provides a protocol for RDF data, it is standardized, supported and recommended by the World Wide Web Consortium (W3C). RDF data is represented as a graph, SPARQL is a graph-querying language and can be used on one or more RDF files in the memory. Once it is stated on where to look for the data, SPARQL can run different types of queries:

- SELECT, which is used to retrieve information, similar to an SQL query
<table>
<thead>
<tr>
<th>Data</th>
<th>Query Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML</td>
<td>XQuery, XPointer, XPath</td>
</tr>
<tr>
<td>RDF</td>
<td>SPARQL, RDQL</td>
</tr>
<tr>
<td>SKOS</td>
<td>SPARQL</td>
</tr>
<tr>
<td>OWL</td>
<td>OWL-QL, SQWRL</td>
</tr>
<tr>
<td>SWRL</td>
<td>SWRL, DLP</td>
</tr>
</tbody>
</table>

**XML**  eXtensible Markup Language

**XQuery, XPointer, XPath**  Query languages for XML data sources

**RDQL**  Query language for RDF in Jena models

**OWL-QL**  Proposed OWL query language by the Joint US/EU ad hoc Agent Markup Language Committee

**SQWRL**  Semantic Query-Enhanced Web Rule Language; pronounced squirrel

**SWRL**  is a rules-language that combines OWL with RuleML

**DLP**  Description Logic Programs

Table 2: Query Languages and list of the abbreviations
PREFIX fr:<http://delicious.com/tag/>

SELECT ?p ?o
FILTER (?p > 0.35) }

Figure 3: SPARQL Query

- CONSTRUCT creates an RDF graph based on RDF input, this can also be used to translate between different ontologies
- ASK is used to find out if a certain query pattern can be used on the queried RDF graph
- DESCRIBE identifies all triples related to a particular object

(Breslin, Passant, Decker, 2009, p. 62)

Modifiers like ORDER and LIMIT can be used as well. With ORDER, the results can be ordered in an ascending or descending manner and LIMIT limits the results to a certain amount of items. FILTER can be used, as the name says, filter out certain results based on their numerical value.

An example SPARQL query like it could be used in the YouReputation code can be seen in Figure 3. As the PREFIX always stays the same, it is shortened using “fr:” instead of “http://delicious.com/tag/”. This prefix is not a real one but was added because RDF data is always saved ad URIs and AllegroGraph, the graph-database used for the project asked for URIs. “?s”, “?p” and “?o” stand for subject, predicate and object. In this query we are looking for the predicate and object to a certain given subject, “apple”. With filter, the predicate, which is a number, is limited to be larger than 0.35. It can be seen later in this thesis, that this predicate is the relevance of the object to the subject.

SPARQL can return results in various formats such as XML, JSON, RDF or HTML.

2.4.2 RDQL

RDQL stands for RDF query language and also retrieves and manipulates data stored in the RDF format. It is, like SPARQL, also a W3C recommendation and can
SELECT variables listing
FROM rdf documents
WHERE patterns
AND filter expressions
USING prefix declaration

Figure 4: RDQL Query Structure

be seen as predating SPARQL query language as it was developed earlier. RDQL was
developed by Hewlett Packard and was derived from SQUISH, an earlier language
(Hutt, 2005, p. 4). Also RDQL has a similar syntax pattern like SQL. Generally, a
RDQL query looks like the one in Figure 4.

- SELECT specifies which variables will be returned
- FROM, here’s where the URI or path to the RDF document is specified
- WHERE specifies the triple pattern to be matched
- AND, the AND part of a query can be used to add Boolean expressions to
  limit query results
- USING is similar to the PREFIX notation in SPARQL, here, URIs can be
  shortened.

The following systems provide RDQL support: Jena (a Java framework for build-
ing Semantic Web applications), RDFStore (Perl API for RDF storage), Sesame
(open source RDF database), PHPXML Classes, 3Store (RDF triple store written
in C), RAP - RDF API for PHP (software package for parsing, querying, manipu-
lating, serializing and serving RDF models).

2.5 Query Language Comparison

The two query languages introduced above will now be compared against each other
to find the one most usable for the YouReputation project. First, the criteria are
listed and then the comparison will deliver a result.
2.5.1 Criteria

The following features help compare query languages:

**Value Comparison and Data Type Support** In order to provide support for value comparison, a query language must exploit any data types inherent in the RDF model.

**Generalized Path Expressions** A syntax for navigating the RDF graph, support for searching a specific pattern, substituting variables in place for a node and for constraining values using Boolean expressions.

**Closure** In RDF, this means that the result of a query operation is a graph and not any other data structure. The result can therefore be queried again.

**Optional Values** Optional sections of a graph can be specified when querying for a matching (similar to the outer join in SQL).

**Aggregate Functions** Functions like MIN, MAX or COUNT

**Advanced Set Operations** (*union, intersection*) This includes the following operations: selection, projection, Cartesian product, set difference, set union.

(Hutt, 2005, p. 4f)

2.5.2 Comparison

Table 3 lists the criteria and shows if SPARQL or RDQL offer the feature or not.

Data type support and path expression syntax are supported by both languages, whereas the third basic feature of query languages, the closure, is not supported by RDQL. Optional sections of a graph can be taken into account when querying with SPARQL, but not with RDQL. Aggregate functions like MIN, MAX or COUNT, as well as set operations can be used in both SPARQL and RDQL.
### SPARQL vs. RDQL

<table>
<thead>
<tr>
<th>Feature</th>
<th>SPARQL</th>
<th>RDQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Type Support</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Path Expressions</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Closure</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Optional Values</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Aggregate Functions</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Advanced Set Operations</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3: Comparison Criteria for Query Languages

#### 2.5.3 Choice & Conclusion

SPARQL will be used for the project implementation. AllegroGraph, the graph database used to store the ontology supports SPARQL queries. Closure, which is supported by SPARQL but not RDQL, can be handy when working with the queries as we might want to query the results to get a more accurate and refined end result.

In the following section of the thesis, the case study example YouReputation will be showing how the PageRank and SPARQL can be used in a Social Semantic Web application.

### 3 Case Example

#### 3.1 Online Reputation Analysis

Every company or person selling products or services of any kind can be sure to be talked about by customers in forums or on comparison websites. This is part of the Social Web, people talk about products and services, about what they bought or what they think about buying. Nowadays it is normal to search for something than one intends to buy online and find out if it’s worth the money or what kind of problems or good experiences they encountered with said item.

Producers and consumers mix and become what can be called a prosumer in today’s World Wide Web. Buyer also sell things, seller also buy things online. For producers it is important to interact with their customers. For example if a customer leaves a bad review on a service he paid for then the producer will want to get in
touch with the consumer and talk about the negative experience and how this can be worked out. Bad reviews online can be read by people all over the world and is worse marketing than good publicity can make up for.

The reputation of a company or their product, or also called corporate reputation, is and has always been an important aspect of success. In the time of the Social Web, next to the offline reputation, the online reputation is what has become very interesting and influential (Uhlmann, 2011, p. 25). Now that consumers interact globally on the Web, much more information and personal experience about a product or service can and will be shared. Online Reputation Analysis is a part of managing and improving the online reputation. The analysis analyzes the current situation.

3.1.1 Reputation as a Term

The reputation of a company is not something that is created by only the customers. Employees, investors, politicians and supplier of the firm also help build this reputation. The media plays an important role too here, as it has influence on all these people. The effects that the reputation has on these groups of people include:

- Customers: Trust in the company and their products, customer loyalty, customer buys the product again, product can be more expensive
- Employees: Easier to find employees for the company, stronger attachment of the employee, lower cost of salaries
- Investors: Higher disposition to buy and keep, lower cost of capital procurement
- Politicians: Support
- Suppliers: Lower acquisition costs, higher attachment

(Uhlmann, 2011, p. 20)

Companies can actively control their reputation, this is called Reputation Management. Online Reputation Management includes monitoring, analyzing and engaging with negative comments about a company’s products, brands, stores and executives on the Internet (Belfort, 2011).
3.1.2 Online Reputation Analysis

The Reputation Analysis is part of the Reputation Management and as that, helps improve the corporate reputation. Online Reputation Analysis is different from Offline Reputation Analysis. Online Reputation Analysis includes everything that is put online by the users of the Web. This includes Social Networks, Wikis, Blogs, Fora and any other form of expressing personal opinions. Google provides two services that help analyze Online Reputation. Google Alerts (http://www.google.com/alerts) combined with Google Reader (http://www.google.com/reader/) can be used as a free reputation analysis tool (Belfort, 2011). With this tool, one can analyze news, blogs, realtime (Twitter, FriendFeed, Facebook, MySpace, indenti.ca, TwitArmy and Jaiku), videos and discussions.

After analyzing, companies can directly interact with the people and, especially when the image is bad, improve the reputation.

3.2 Introduction to the Prototype

The website YouReputation is a fuzzy online reputation analysis framework, short FORA framework, by the University of Fribourg and the NUS. The goal is to help people, companies and organizations to improve their online reputation management.

On YouReputation the goal is to give the producers or companies a way to analyze their Online Reputation in blogs, forums and social websites where consumers talk about their product. It is a search engine designed to help people find negative comments and reviews in order for the producer to be able to interact and improve the own reputation.

In order for this to be possible, there has to be a connection to the Semantic Web. The search engine has to be able to differentiate between different products and services and find them in the Social Web. An RDF store helps store the ontology. For the FORA framework, AllegroGraph by www.franz.com has been chosen.

AllegroGraph is a graph database for Semantic Web applications. It stores data in form of triplets which can be queried using SPARQL, RDFS++ or Prolog. A triplet includes a subject, predicate and object (s, p, o). As the comparison in the first part of the thesis showed, it makes sense to use SPARQL to query RDF data.
3.3 Functionality of the Prototype

On YouReputation a search term is entered and then, AllegroGraph is accessed and queried with a SPARQL query. It gives back the search terms that are similar to the query. The percentage of this similarity can be changed by the user. The result terms are displayed in a topic map and on the right side of the page delicious links for the query result are displayed as can be seen in Figure 5.

**Figure 5: The website YouReputation**

The distance of a term, or otherwise called tag, shows how relevant it is to the center. The tags around the center are arranged in a spiral manner so it can easily be seen which is the closest match.
4 Practical Section

4.1 Search Result Ranking for the Social Semantic Web

Now we will have a look at the practical part of this thesis. It was planned to be an implementation of a ranking algorithm for YouReputation. After certain unforeseen changes and adaptions, the result is now ranked by delicious and the retrieval, using SPARQL, is implemented and working.

4.2 Annotations to the Code

Using fuzzy clustering, as explained in (Kolly, 2011), the 517 tags stored in AllegroGraph have been grouped together. There are 10 center tags. For each of the 517 tags the distance to the ten centers is stored and can be accessed. This is the current situation when working on the project. The numbers 10 and 517 often used in this annotation can change when the ontology saved in AllegroGraph changes.

In the following paragraphs, the functionality of the Hypertext Preprocessor (PHP) code (see appendix) is explained in more detail.

4.2.1 User Input, Database Connection and Query

In order to get a result, the user has to enter a query which is then, after being connected to AllegroGraph, is searched for using a SPARQL query.

Handling User Input  First, the user enters a keyword and sets a percentage value for how relevant the proposed tags should be to this keyword. These two values, keyword and percentage, are stored in a variable and the percentage is divided by a hundred to obtain a value between zero and one.

Prepare Connection to AllegroGraph  The connection to AllegroGraph is done using HTTP protocol. First, login information (username and password) is encoded with base64 encryption. Connection details are saved in an array. The array includes information such as:

- the method: GET, as we are getting information from AllegroGraph using HTTP protocol
SELECT ?s ?p ?o
WHERE { ?s ?p ?o }

Figure 6: SPARQL Query to get data from AllegroGraph

• the header states that we are using HTTP protocol 1.1

• the return type is JSON, an array of arrays of strings with the inner arrays having three or four elements each, other possible return types are: text/plain, application/rdf+xml, text/x?nquads, application/trix, text/turtle

• basic authorization is done with the login data saved beforehand

• after use, the connection is closed

Connect to AllegroGraph and Query it using SPARQL  First, the connection is created with the array saved previously.

The SPARQL query is of the most simple kind, see Figure 6, because of two major problems:

1. RDF data is stored in form of a URI and it is not possible to treat “www.delicious.com/0.358893” like a number and use the FILTER method to get only the desired results

2. because we have ten centers, and the only distances saved for each tag are the distances to these centers, we cannot find the distance or relevance of one tag to another when both are not centers (which is usually the case)

The whole database has to be retrieved from AllegroGraph and saved into an array for further use.

The result of the query with 517 tags, each one with the center distances to 10 centers, comes from the AllegroGraph repository called “Delicious517Tags” and is stored as an array in a variable.

4.2.2 Preparing the Result for Computations

The result has to be prepared and rearranged for further use.
Creating Arrays for the Retrieved Data and Cutting the URIs  For further use, this first big array is split up into three arrays, one with the tags, one with the relevances and one with the centers. The URIs are cut out, so that instead of “www.delicious.com/tags/webdesign”, the tag is saved as “webdesign”.

Remodeling the Arrays  Now the three arrays for tags, relevances and centers each still have too many elements. They have to be remodeled so the tags array has its 517 tags, the relevances array has the 517 times 10 relevances and the centers array has the ten centers in it.

Using the function “array_unique()” the centers are easily reduced to be the ten centers only once instead of 517 times ten centers.

With “array_chunk” it is possible to create an array of inner arrays. So the relevances array now has 517 elements, each of them being an array of 10 relevances for 10 centers. Of course, as mentioned before, if the amount of centers changes, these inner arrays have also a different amount of center distances in them.

The tags array is reduced as well so it only has each tag once instead of ten times.

Preparation to Calculate Close Tags  To find out which tags are close to the keyword, the idea is simple: a calculation is done to find out which tag has similar center distances to the keyword.

If, for example, we only have two centers A and B, and three tags 1, 2 and 3, if tag 1 has distance 50 to A and 70 to B, tag 2 has distance 20 to A and 60 to B and tag 3 has distance 45 to A and 80 to B, it is clear that tag 1 and tag 3 are closer to each other than to tag 2.

In this sense, using the percentage given by the user, two arrays are created, one called “fuzzy_min” and the other called “fuzzy_max” to save the minimal and maximal values a certain center distance is allowed to have in order to be considered close to the keyword. The relevance array of the keyword is iterated and the value of each item is decreased and increased by the user-given percentage.

If the value of the keyword is zero, then the tag is a center tag and we have to handle it differently as decreasing zero by a percentage makes no sense, the minimal value is set to be zero and the maximal value is set to be the value minus half the
percentage. This can sound a bit strange but after testing, the results (amounts of tags returned) make more sense than with any other calculation.

4.2.3 Find Close Tags

After these preparations, the arrays can be queried to find the close tags.

Calculate Tags with Similar Center Distances A loop in a loop checks if each of the 517 inner arrays of the relevance array store values that are similar to the value in the keyword relevance array. So the relevance of a tag to the first center is taken and it is checked if it is larger than the keyword-relevance to the first center decreased by the percentage value and also if it is smaller than the keyword-relevance to the first center increased by the percentage value (as stored in “fuzzy_max”).

If both are true, the tag is added to a new array. Now each tag could be in this array 10 times, if all its center distances fit the previous test. The more often a tag is in this array, the more center tags fit the specifications and the closer it is to the keyword.

Because the keyword itself is tested as well, and is listed ten times in the array, it has to be removed for further processing of the results.

Selecting and Arranging the Tags New arrays are created:

- one to count how many times a tag has a mutual center with the keyword
- another to find the keys of the mutual centers
- thirdly, one to save the values of the tags that have mutual centers with the keywords
- “relevant_tags_prev” saves the effectively relevant tags, they are found looping through the arrays in order to find only the tags that have at least eight mutual centers (or, if the amount of centers changes, two centers less than the maximum).

A problem that occurred during trying is that the centers are not in the list of relevant tags. They need to be checked and added as well if necessary and if close enough to the keyword tag. Here it turned out to make most sense to add a center if it has more relevance than half of the inverse of the percentage.
4.2.4 Center Distances

The center distances now have to be calculated. The following section explains how this is done.

Calculating Distances from Tags to Keyword  At first, it was not necessary
to calculate the distances, but later on in the project, these distances turned out
to be necessary. If they would have been calculated earlier on, the calculation of
close tags would have been done differently, by only calculating distances and then
finding out which tags were close.

Finding the distances (lines 273ff in the appendix):

- for a counter, if it is smaller than the number of relevant tags
- an internal array is created, it holds the center distances of the first relevant
tag
  - loop through the center distances of the tags
  - subtract the center distance of the keyword
  - calculate the square of the subtraction
  - put the value into the array

- calculate the sum of all, in our current case 10, values obtained in the loop
  above

- take the square root of the sum, this returns the distance between the two tags

4.2.5 Handling and Returning the Result

In the end, the result is processed and returned, how this is done is explained in the
following paragraphs.

Process the Result and Return it  The results have to be processed and re-
turned in JSON format. For this, a loop is done to save the relevant tags and their
corresponding distances. The result is given its final form with inner arrays that
hold names (“tag” and “distance”) for their elements and the keyword is added to
the front of the result array when encrypting it all to JSON. To the end of the result
data, a list of the centers is added because they need to be visualized differently. If
the percentage given by the user is zero, only the keyword is returned.

The returned data could look like in Figure 7.

After that, the result is returned to be processed further. The keyword and
relevant tags are searched for on Delicious (www.delicious.com). Getting the links
from Delicious is quite difficult as the API does not support this feature. The
ranking is currently done by Delicious.

In a further step, the relevant tags could be searched on several search engines
and the results could be ranked using PageRank.
5 Conclusion & Outlook

In this final part the thesis will be drawing a conclusion of the previous chapters and give an outlook on a possible future of the Web.

Combining the Social Web and the Semantic Web to the Social Semantic Web opens many new possibilities of using, storing and retrieving information. Information that not only makes sense to humans but to machines as well helps us make more out of the Web. It helps create a collective intelligence where all people connected to the Internet are able to contribute. Computers can help organize and categorize the large amount of information that currently exists multiple times on several websites. Search engines become intelligent. IR will “know” what the user wants or show him what he could want but does not know about yet.

This thesis showed that it is not easy to find the perfect tools to build Social Semantic Web applications. Some are better for one kind of applications and some are better for others, all have their positive and negative characteristics and perform differently. SPARQL was found to be most useful for the project described in the last section, but while working, it turned out to be not usable with numbers in our current example. It was difficult to work around this fact.

In a future step, the part of the project described here could be extended to include several search engines and the ranking could by done by a ranking algorithm like PageRank for example. The goal of YouReputation is to become a website where users can find their company or product and especially the customers opinion about it. Users can start interacting with the customers in order to get a better reputation.

Semantic Web technologies can bring many benefits to social networks and websites, but the Social Semantic Web is still a vision instead of reality (Breslin, Passant, Decker, 2009, p. 281). Sites are still disjoint, information still exists on multiple pages but, as we have seen, the Web is evolving and changing all the time and the technologies are here. As the need of more Social Semantic Web languages becomes more obvious, the current technologies evolve. There exists a Fuzzy SPARQL called f-SPARQL now (. This includes fuzziness and can work with it.

People can share, exchange, trade, sell and meet online, the world seems smaller. Soon, appliances in our houses can be connected to each other and will be controlled over the Web. Before leaving work, one can turn on the oven at home by sending
it a signal from the mobile phone. Tim Berners-Lee pointed out that the future of
the Web as we know it lies in mobile phones, in the future smartphones.

Thank You  In this last paragraph, I would like to thank Edy Portmann for his
support, endurance and patience during the process of this bachelor thesis. Thank
you, Edy. Also I would like to thank Pascal Burkhard and Sandro Kolly for their
time and effort when working on the interfaces between our parts of the practical
part.
Literature

**Belfort, 2011** Belfort, F.: Social Media Monitoring: Online Reputation Management on the Social Web; Bachelor Thesis at the University of Fribourg, 2011


**Fischerländler, Schoeberichts** Fischerländler, S., Schoeberichts, M.: Grundlagen der aktuellen Ranking-Algorithmen; [http://www.suchmaschinentricks.de/ranking/grundlagen](http://www.suchmaschinentricks.de/ranking/grundlagen) accessed in November 2011

**Franceschet, 2010** Franceschet, M.: PageRank: Stand on the shoulders of giants; Departement of Mathematics and Computer Science, University of Udine, 2010


**Hutt, 2005** Hutt, K.: A Comparison of RDF Query Languages, 21st Computer Science Seminar, Rensselaer Polytechnic Institute, 2005


Mäkeläinen, 2005 Mäkeläinen, S.: Social Semantic Web; University of Helsinki, 2005


Meier, Reich, 2011 Meier, A., Reich, S.: Communitys im Web; HMD, Heft 280, August 2011

Pan, Stamou, Taylor, 2008 Pan, J., Stamou, G., Taylor, S.: Scalable Querying Services over Fuzzy Ontologies; April 2008


Appendix

The Code

<!-- Author: Andrea Liechti

This implementation is the practical section of my bachelor thesis. It's part of
a big project: yourreputation.org and it's purpose is to connect to allegrograph
(where the ontology is stored) and read the triplets stored there. After that,
the information obtained is used to find out which tags are close to a keyword
entered by the user. These tags and their distance from the keyword are returned
to be used in the graphical display of the tag cloud.

(Because it was formatted differently, the comments are under the code line
they're commenting.) -->

<?php

/********************************************
* handling the user input
********************************************/

$keyword = trim(strtolower($_GET['keyword']));
//getting and saving the keyword that the user entered

$percent = $_GET['percentage'];
//getting and saving the percentage

$percent = $percent / 100;
//divide the percentage by 100 (will be used later)

/********************************************
* connection details to connect to allegrograph using HTTP protocol
********************************************/

$login = base64encode("allegro:e67HJ3q");
//encode the allegrograph username and password

$connection = array(
    'http'=>array(
        //http protocol
        'method'=>'GET',
        //getting information
        'header'=> "HTTP/1.1\r\n" .
        //http header
        "Accept: application/json\r\n" .
        //return type is application/json – an array of arrays of strings,
with the inner arrays having 3 or 4 elements each

other response types are: text/plain; application/rdf+xml;
//text/x-nquads; application/trix; text/turtle
"Authorization: Basic $login

" using basic authorization with the encoded login data encoded above
"Connection: close

" after using it, the connection is closed

);}

/*******************************************************************************/
start connection and querying allegrograph using sparql
/*******************************************************************************/

$context = stream_context_create($connection);
//connect using details above
//the sparql query – because it’s not possible to calculate and query
//with numbers, it’s necessary to query all
$result = file_get_contents("http://www.youreputation.org:8081/repositories/Delicious517Tags?
query={$query", null, $context};
//with the connection to allegrograph,
//the repository "Delicious517Tags" is accessed and the query above executed

/*******************************************************************************/
* saving the data in arrays
/*******************************************************************************/

$tags = array();
//array for the tags
$relevances = array();
//array for the relevances
$centers = array();
//array for the centers (won’t be used as for now)
$xml = simplexml_load_string($result);
//load the string as simplexml element
foreach($xml->results->result as $result) {
    //going through the result one by one
    $tag = strtolower($result->binding[0]->uri);
    //each tag is saved in the $tag variable until the next iteration of the loop
$relevance = $result->binding[1]->uri;

// each relevance is saved until the next iteration of the loop
$center = strtolower($result->binding[2]->uri);

// and so is the center
$tag = substr($tag, 25);

// cutting the uri that was part of the allegrograph triplet
$relevance = substr($relevance, 21);

// cutting the uri
$center = substr($center, 25); // cutting the uri

array_push($tags, $tag);
array_push($relevances, $relevance);
array_push($centers, $center);

// adding the center to the centers array

} // remodeling the arrays

centers = array_unique($centers);

// an array that only contains the centers, in our current case 10

numberofc = count($centers);

// in case something changes and we have a different amount of centers, the number of centers is adaptable

rel = array_chunk($relevances, $numberofc);

// creating an array in an array so that all x relevances (to x centers) for one tag are an array (length x) in an array (length 517)

tag = array();

// creating an array for the tags
foreach($tags as $i => $eachtag) {

// looping through the currently 5170 elements long array

if($i % $numberofc == 0) {

// and picking every tenth (amount of centers) element

array_push($tag, $eachtag);

// so that we have each tag only once, giving us an array of 517 (for 517 tags, amount can change) elements

}

}
$key = array_search($keyword, $tag);

//get the key of the keyword entered by the user
$fuzzy_min = array();
//creating array for the minimal values
$fuzzy_max = array();
//creating array for the maximal values
foreach($rel[$key] as $value) {

//iterating through the relevances array that holds the distances of the keyword to the ten centers
if ($value == 0) {

//the value here is 0.000000 if the tag is one of the centers calculated by the algo
$val_min = 0;

//in that case we can’t take a percentage of the value because that’s 0 and would return nothing but the center tag itself
$val_max = $value + $percent/2;

//for that reason, we take 0 as the lower boundary and 0 plus the percentage for the maximum value and divide the percentage by 2 so we don’t have hundreds of results
}
else if ($value == 1) {

//also the case with centers
$val_min = $value - $percent/2;

//here, the minimal value is the value minus half the percentage
$val_max = 1;

//and the maximum is 1
}
else {

$val_min = $value - $value * $percent;

//for all other tags: calculating distance to center minus the percentage entered by the user
$val_max = $value + $value * $percent;

//calculating the distance to center plus the percentage entered by the user
}
array_push($fuzzy_min,$val_min);

//adding the minimal values a tag has to have to be considered close to another tag
array_push($fuzzy_max,$val_max);
//adding the maximal values a tag has to have to be consideret close to another tag

/******************
* calculate tags with similar center distances
***************/
$fuzzy_tags = array();
//an array to save the tags if their center distances are similar to the respective center distance
//of the keyword
$counter = 0;
//a counter in order to have the key to the elements that pass the if loop
foreach($rel as $value) {
    //loop through the relevances array 0 to 516 (in the current case)
    $counter2 = 0;
    //counter to have the key of the inner array
    foreach($value as $value2) {
        //loop through all inner arrays
        if(($fuzzy_min[$counter2] <= $value2) && ($value2 <= $fuzzy_max[$counter2])) {
            //check if the center distance is larger than the minimal value calculated above &&
            //check if it’s smaller than the maximal value
            array_push($fuzzy_tags,$tag[$counter]);
            //if both are true, add the tag to the array which now has 472 elements. if i search for
            //sport, sport is 10 times in this array,
        }
        //something that has 6 center distances that fit the if loop is 6 times in the array
        $counter2++;
    }
    //update the counter
    $counter++;
}
//update the counter
foreach($fuzzy_tags as $i => $value) {
    //loop through the fuzzy_tags array
    if($value == $keyword) {
        //to check for the keyword
        unset($fuzzy_tags[$i]);
        //and delete the 10 elements containing it, as it does not need to be part of the tags
    }
}
selecting and arranging the tags

$mutual\_centers = \text{array}();$

//an array

$mutual\_centers = \text{array\_count\_values}(\$\text{fuzzy\_tags});$

//to count how many times a tag has a mutual center with the keyword

$mutual\_center\_vals = \text{array}();$

//an array

$mutual\_center\_vals = \text{array\_keys}(\$mutual\_centers);$  

//to find the keys of the mutual centers

$mutual\_center\_amount = \text{array}();$

//an array

$mutual\_center\_amount = \text{array\_values}(\$\text{mutual\_centers});$

//to save the values of the tags that have mutual centers with the keyword

$relevant\_tags\_prev = \text{array}();$

//an array to save the effectively relevant tags

$\text{center\_counter} = \$\text{numberofc};$

//maximum 10 of 10 centers can be mutual

do {
  $\text{tag\_keys} = \text{array\_keys}(\$\text{mutual\_center\_amount},\$\text{center\_counter});$

  //create an array with the tags and the amount of center distances that are similar to the
  //center distances of the keyword

  foreach($\text{tag\_keys} \text{as} \$\text{value}) {
    //iterate through it

    array\_push($\text{relevant\_tags\_prev},\$\text{value});
    //pushing the values to the array created before the loops

  }

  $\text{center\_counter}--;$

  //update the counter

} while($\text{center\_counter} > \$\text{numberofc} - 2);

//do all this only for tags that have a lot of mutual centers

$relevant\_tags = \text{array}();$

//array for the relevant tags (keys of the $\text{tag} \text{array}$)
foreach($relevant_tags_prev as $value) {
    //for each element of the relevant tags array containing only the tags that had mutual centers
    $reltag = array_search($mutual_center_vals[$value],$tag);
    //search the tags in the larger arrays
    array_push($relevant_tags,$reltag);
    //and put the keys in the relevant tags array
}

$include_center = array_search($keyword,$tag);
//to include centers (that may be further away from the tags, we search for the keyword in the
tags array
$incl_ctr_rel = array();
//an array is created
$incl_ctr_rel = $rel[$include_center];
//and the center relevances of the keyword are saved
foreach($incl_ctr_rel as $i => $value) {
    //then iterated
    if($value > 1 -$percent/2 && $centers[$i] != $keyword) {
        //and if a center is not the keyword but has more relevance than half of the inverse of the
        percentage
        $incl_ctr = array_search($centers[$i],$tag);
        //then the center is searched for in the tag array
        array_push($relevant_tags,$incl_ctr);
        //and added to the relevant tags array
    }
}

/*******************************************************************************
* calculating distances from tags to keyword
*******************************************************************************/
$data = array();
//array
foreach($relevant_tags as $value) {
    //find the relevant tags
    array_push($data,$tag[$value]);
    //put them in the array
}
$keydist = array();
//array
foreach($rel[$key] as $value) {
    //iterate through the center distances of the keyword
}
array_push($keydist,$value);

//put them in the array

$centerdists = array();
//new array

foreach($relevant_tags as $value) {
    //find the center distances needed for the calculation

    array_push($centerdists,$rel[$value]);
    //push them into the array
}

$added = array();
//array for the results of the distance calculations

$sumvalue = 0;
//variable for the distances

for($ct_ar = 0; $ct_ar < count($centerdists); $ct_ar++) {
    //for a counter, if it is smaller than the number of relevant tags

    $internal_array = $centerdists[$ct_ar];
    //an internal array is created, it holds the center distances of the first relevant tag

    $inner_counter = 0;
    //a counter is created

    $addition = array();
    //and a new array

    foreach($internal_array as $value) {
        //loop through the center distances of the tags

        $dist_value = $value - $keydist[$inner_counter];
        //subtract the center distance of the keyword from the center distance of the tag

        $dist_value = pow($dist_value,2);
        //calculate the square of the substraction

        array_push($addition,$dist_value);
        //put the value into the array

        $inner_counter++;
        //update the counter
    }

    $sumvalue = array_sum($addition);
    //calculate the sum of all, in our current case 10, values obtained in the foreach loop above

    $sumvalue = sqrt($sumvalue);
    //take the square root of the sum, this gives us the distance between the two tags

    array_push($added,$sumvalue);
    //put the distance in an array
}

48
process the result and return it

$data_dist = array();
$array
$count = 0;
$counter
foreach($relevant_tags as $value) {
//go through relevant tags
$tagdata = array($added[$count],$tag[$value]);
//make an array with the relevant distance and the corresponding tag
array_push($data_dist,$tagdata);
//put this small array in a larger array holding all distances and tags
$count++;
//update the counter
}
$data_result = array();
//create a new array
foreach ($data_dist as $row) {
//go through the array with the distances and the tags
$datarr = array("distance"=>$row['0'],"tag"=>$row['1']);
//name the key for the distances "distance" and the key for the tags "tag"
array_push($data_result,$datarr);
//add them all up in a new array
}
$result_tags = json_encode(array("keyword"=>$keyword,"data"=>$data_result,"centers"=>$centers));
//encode and format the result
if($percent == 0) {
//if the percentage is 0
   echo json_encode(array("keyword"=>$keyword));
//the keyword is still returned
} else {
   echo $result_tags;
//if the percentage is something other than 0, the result is returned
}