Ranking the Linked Data: The Case of DBpedia

Roberto Mirizzi[1], Azzurra Ragone[1,2],
Tommaso Di Noia[1], and Eugenio Di Sciascio[1]

[1] Politecnico di Bari – Via Orabona, 4, 70125 Bari, Italy
mirizzi@deemail.poliba.it, {ragone,dinoia,disciacio}@poliba.it
[2] University of Trento – Via Sommarive, 14, 38100 Povo (Trento), Italy
ragone@disi.unitn.it

Abstract. The recent proliferation of crowd computing initiatives on the web calls for smarter methodologies and tools to annotate, query and explore repositories. There is the need for scalable techniques able to return also approximate results with respect to a given query as a ranked set of promising alternatives. In this paper we concentrate on annotation and retrieval of software components, exploiting semantic tagging relying on Linked Open Data. We focus on DBpedia and propose a new hybrid methodology to rank resources exploiting: (i) the graph-based nature of the underlying RDF structure, (ii) context independent semantic relations in the graph and (iii) external information sources such as classical search engine results and social tagging systems. We compare our approach with other RDF similarity measures, proving the validity of our algorithm with an extensive evaluation involving real users.

1 Introduction

The emergence of the crowd computing initiative has brought on the web a new wave of tools enabling collaboration and sharing of ideas and projects, ranging from simple blogs to social networks, sharing software platforms and even mashups. However, when these web-based tools reach the "critical mass" one of the problem that suddenly arises is how to retrieve content of interest from such rich repositories. As a way of example, we can refer to a platform to share software components, like `ProgrammableWeb`[1], where programmers can share APIs and mashups. When a user uploads a new piece of code, she tags it so that the component will be later easily retrievable by other users. Components can be retrieved through a keywords-based search or browsing accross *categories*, *most popular* or *new updates*. The limits of such platforms, though very popular and spread out on the entire web, are the usual ones related to keywords-based retrieval systems, e.g., if the user is looking for a resource tagged with either *Drupal* or *Joomla!*[2], the resources tagged with *CMS* (*Content Management System*) will not be retrieved. For example, in ProgrammableWeb, APIs as *ThemeForest* and

[1] `http://www.programmableweb.com`
[2] `http://www.drupal.org`, `http://www.joomla.org`

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 337–354, 2010.

Ecordia[3] are tagged with *CMS* but not with *Drupal* nor *Joomla*, even if in their abstracts it is explicitly written that they are available also for the two specific CMSs.

An effective system should be able to return also *approximate* results w.r.t. the user's query, results ranked based on the *similarity* of each software components to the user's request. Referring to the previous example, it means that the two mentioned APIs should be suggested as relevant even if the exact searched tag is not present in their description, due to their similarities with the query.

Another issue stricly coupled with the keyword-based nature of current tagging systems on the web is synonymy. Different tags having the same meaning can be used to annotate the same content. `Faviki`[4] is a tool for social bookmarking that helps users to tag documents using `DBpedia` [2] terms extracted from Wikipedia. Although it is a good starting point to cope with synonymy, it does not solve the problem of *ranking* tags w.r.t. a query. Moreover it does not provide the user with any suggestion of tags related to the ones selected during the annotation phase, e.g., if the user tags a page with *Drupal*, the tool does not suggest to tag the page with *CMS* too.

Partially inspired by `Faviki`, in this paper we propose a new hybrid approach to rank `RDF` resources within `Linked Data` [1], focusing in particular on `DBpedia`, which is part of the Linked Data Cloud. Given a query (tag), the system is able to retrieve a set of ranked resources (e.g., annotated software components) semantically related to the requested one. There are two main relevant aspects in our approach: (1) the system returns resources within a specific context, e.g., *IT*, *Business*, *Movies*; (2) the final ranking takes into account not only `DBpedia` links but it combines the `DBpedia` graph exploration with information coming from external textual information sources such as web search engines and social tagging systems.

A system able to compute a ranking among `DBpedia` nodes can be useful both during the annotation phase and during the retrieval one. On the one hand, while annotating a resource, the system will suggest new tags semantically related to the ones already elicited by the user. On the other hand, given a query formulated as a set of tags, the system will return also resources whose tags are semantically related to the ones representing the query. For instance, if a user is annotating an API for ProgrammableWeb with the tag *CMS* (which refers to `DBpedia` resource `http://dbpedia.org/resource/Content_management_system`), then the system will suggest related tags as *Drupal*, *Joomla* and *Magento* (each one related to their own `DBpedia` resource).

Main contributions of this work are:

– A novel *hybrid* approach to rank resources on `DBpedia` w.r.t. a given query. Our system combines the advantages of a *semantic-based* approach (relying on a `RDF` graph) with the benefits of *text-based* IR approaches as it also exploits the results coming from the most popular *search engines* (Google, Yahoo!, Bing) and from a popular *social bookmarking system* (Delicious). Moreover,

[3] `http://www.programmableweb.com/api/themeforest|ecordia`
[4] `http://www.faviki.com`

our ranking algorithm is enhanced by *textual* and *link analysis* (abstracts and wikilinks in `DBpedia` coming from Wikipedia).
- A *relative* ranking system: differently from PageRank-style algorithms, each node in the graph has not an importance value per se, but it is ranked w.r.t. its neighbourhood nodes. That is, each node has a different importance value depending on the performed query. In our system we want to rank resources w.r.t. a given query by retrieving a ranked list of resources. For this reason we compute a weight for each mutual relation between resources, instead of a weight for the single resource, as in PageRank-style algorithms.
- A back-end system for the semantic annotation of web resources, useful in both the *tagging* phase and in the *retrieval* one.
- An extensive evaluation of our algorithm with real users and comparison w.r.t. other four different ranking algorithms, which provides evidence of the quality of our approach.

The remainder of the paper is structured as follows: in Section 2 we introduce and detail our ranking algorithm *DBpediaRanker*. In Section 3 we present a prototype that highlight some characteristics of the approach. Then, in Section 4, we show and discuss the results of the experimental evaluation. In Section 5 we discuss relevant related works. Conclusion and future work close the paper.

2 DBpediaRanker: RDF Ranking in DBpedia

In a nutshell, *DBpediaRanker*[5] explores the `DBpedia` graph and queries external information sources in order to compute a *similarity value* for each pair of resources reached during the exploration. The operations are performed offline and, at the end, the result is a weighted graph where nodes are `DBpedia` resources and weights represent the similarity value between the two nodes. The graph so obtained will then be used at runtime, (i) in the annotation phase, to suggest *similar* tags to users and (ii) in the retrieval phase, to retrieve a list of resources, ranked w.r.t. a given query.

The exploration of the graph can be limited to a specific *context*. In our experimental setting we limited our exploration to the IT context and, specifically, to programming languages and database systems, as detailed in Section 2.3.

For each node in the graph a depth-first search is performed, stopped after a number of n hops, with n depending on the context. The exploration starts from a set of **seed nodes** and then goes recursively: in place of seed nodes, at each step the algorithm identifies a number of **representative nodes** of the context, i.e., popular nodes that, at that step, have been reached several times during the exploration. Representative nodes are used to determine if every node reached during the exploration is in the context and then should be further explored in the next step. This is done computing a similarity value between each node and representative ones – if this similarity value is under a certain threshold, the node will not be further explored in the subsequent step.

[5] For a more detailed description of the system the interested reader can refer to `http://sisinflab.poliba.it/publications/2010/MRDD10a/`

The similarity value is computed querying *external information sources* (search engines and social bookmarking systems) and thanks to *textual* and *link analysis* in `DBpedia`. For each pair of resource nodes in the explored graph, we perform a query to each external information source: we search for the number of returned web pages containing the labels of each nodes individually and then for the two labels together (as explained in Section 2.2). Moreover, we look at **abstracts** in Wikipedia and **wikilinks**, i.e., links between Wikipedia pages. Specifically, given two resource nodes a and b, we check if the label of node a is contained in the abstract of node b, and vice versa. The main assumption behind this check is that if a resource name appears in the *abstract* of another resource it is reasonable to think that the two resources are related with each other. For the same reason, we also check if the Wikipedia page of resource a has a *(wiki)link* to the Wikipedia page of resource b, and vice versa.

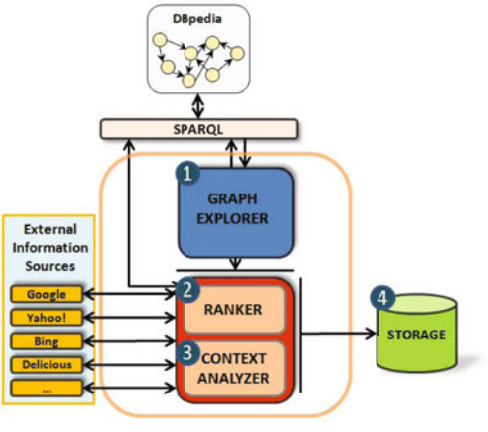

Fig. 1. The ranking system *DBpediaRanker*

In the following we will describe all the components of our system, whose architecture is sketched in Figure 1. The main data structure we use in the approach contains information about `DBpedia` resources[6] reached during the exploration. Hence, for each reached resource an associated data structure r is defined as:

$$r = \begin{cases} URI : \text{a } \texttt{DBpedia} \text{ URI;} \\ hits : \text{how many times the URI has been reached during the exploration;} \\ ranked : \text{Boolean value representing if the URI has already been ranked with respect to its neighbors;} \\ in\_context : \text{Boolean value stating if the URI has to be considered within the context or not;} \end{cases}$$

As the exploration starts from the seed nodes, a global variable $\mathcal{R}$ is initialized with the set of *seed nodes* and then it is further populated with other nodes reached during the graph exploration (see Algorithm 1 in the Appendix). Seed nodes must belong to the context to explore and are selected by domain experts.

[6] From now on, we use the words *URI*, *resource* and *node* indistinctly.

The algorithm explores the DBpedia graph using a depth-first approach up to a depth of MAX_DEPTH (see Section 2.1).

2.1 Graph Explorer

This module queries DBpedia via its SPARQL endpoint[7]. Given a DBpedia resource, the explorer looks for other resources connected to it via a set of predefined properties. The properties of DBpedia to be explored can be set in the system before the exploration starts. In our initial setting, we decided to select only the SKOS[8] properties skos:subject and skos:broader[9]. Indeed, these two properties are very popular in the DBpedia dataset. Moreover, we observed that the majority of nodes reached by other properties were also reached by the selected properties, meaning that our choice of skos:subject and skos:broader properties does not disregard the effects of potentially domain-specific properties.

Given a node, this is explored up to a predefined distance, that can be configured in the initial settings. We found through a series of experiments that, for the context of programming languages and databases, setting $MAX\_DEPTH = 2$ is a good choice as resources within two hops are still highly correlated to the root one, while going to the third hop this correlation quickly decreases. Indeed, we noticed that if we set $MAX\_DEPTH = 1$ (this means considering just nodes directly linked) we lost many relevant relation between pairs of resources. On the other hand, if we set $MAX\_DEPTH > 2$ we have too many non relevant resources.

In order to find the optimal value for MAX_DEPTH, we initially explored 100 seed nodes up to a $MAX\_DEPTH = 4$. After this exploration was completed, we retrieved the top-10 (most similar) related resources for each node (see Section 2.2). The results showed that on the average the 85% of the top-10 related resources where within a distance of one or two hops. The resources two hops far from the seeds where considered as the most relevant the 43% of times ($\sigma = 0.52$). On the contrary the resources above two hops were rarely present among the first results (less than 15% of times). In figure 2 the average percentage of top-10 related resources w.r.t. to the distance from a seed (MAX_DEPTH) is shown.

The exploration starts from a node *root*. Given a DBpedia node *root* and a maximal depth to be reached, this module browses (using a depth-first approach) the graph from *root* up to a number of hops equal to MAX_DEPTH (see Algorithm 2 in the Appendix). For each node u discovered during the exploration, we check if u is relevant for the context and computes a similarity value between *root* and u. Such value is computed by the module *Ranker* as detailed in Section 2.2. As we said before, given a resource u, during the exploration of the RDF graph we analyze only the properties/links skos:subject and skos:broader for which u is either rdf:subject or rdf:object.

[7] http://www.w3.org/TR/rdf-sparql-query/

[8] http://www.w3.org/2004/02/skos/

[9] skos:subject has been recently deprecated in the SKOS vocabulary. Nevertheless, in DBpedia it has not been replaced by its corresponding dcterms:subject.

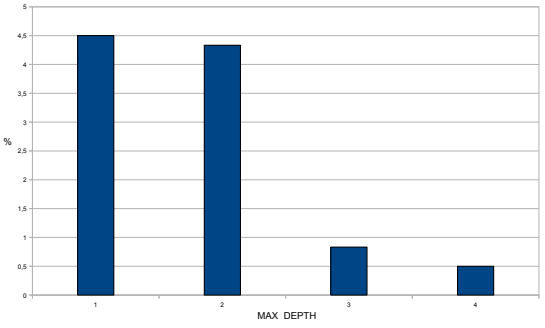

Fig. 2. Evaluation for MAX_DEPTH. It represents the average percentage (y axis) of the top-10 resources related to 100 seeds within a distance of 1 to 4 hops (x axis).

2.2 Ranker

This is the core component of the whole system. Given two resources u_1 and u_2 in the same graph-path, it compares how much they relate with each other exploiting information sources external to `DBpedia` such as search engines and social tagging systems (see Algorithm 3 in the Appendix).

The aim of this module is to evaluate how strong a semantic connection is between two `DBpedia` resources using information taken from external sources. In our current implementation we consider as external sources both web search engines (Google, Yahoo! and Bing) and social tagging systems (Delicious), plus Wikipedia-related information contained in `DBpedia`. Given two `DBpedia` resources u_1 and u_2, we verify how many web pages contain (or have been tagged by) the value of the `rdfs:label` associated to u_1 and u_2. Then we compare these values with the number of pages containing (or tagged by) both labels. We select more than one search engine because we do not want to bind the result to a specific algorithm of a single search engine. Moreover, we want to rank a resource not only with respect to the popularity of related web pages on the web, but also considering the popularity of such resources among users (e.g., in Delicious). In this way we are able to combine two different perspectives on the popularity of a resource: the one related to the words occurring within web documents, the other one exploiting the social nature of the current web. Through formula (1) we evaluate the related similarity of two resources u_1 and u_2 with respect to an external information source $info\_source$.

$$sim(u_1, u_2, info\_source) = \frac{p_{u_1,u_2}}{p_{u_1}} + \frac{p_{u_1,u_2}}{p_{u_2}} \tag{1}$$

Given the information source $info\_source$, p_{u_1} and p_{u_2} represent the number of documents containing (or tagged by) the `rdfs:label` associated to u_1 and u_2 respectively, while p_{u_1,u_2} represents how many documents contain (or have been tagged by) both the label of u_1 and u_2. It is easy to see that the formula is symmetric and the returned value is in $[0, 2]$. *Ranker* does not use only external information sources but exploits also further information from `DBpedia`.

In fact, we also consider Wikipedia hypertextual links mapped in `DBpedia` by the property `dbpprop:wikilink`. Whenever in a Wikipedia document w_1 there is a hypertextual link to another Wikipedia document w_2, in `DBpedia` there is a `dbpprop:wikilink` from the corresponding resources u_1 and u_2. Hence, if there is a `dbpprop:wikilink` from u_1 to u_2 and/or vice versa, we assume a stronger relation between the two resources. More precisely, we assign a score equal to 0 if there are no `dbpprop:wikilinks` between the two resources, 1 if there is a `dbpprop:wikilink` just in one direction, 2 if both resources are linked by `dbpprop:wikilink` in both directions. Furthermore, given two resources u_1 and u_2, we check if the `rdfs:label` of u_1 is contained in the `dbpprop:abstract` of u_2 (and vice versa). Let n be the number of words composing the label of a resource and m the number of words composing the label which are also in the abstract, we also consider the ratio $\frac{m}{n}$ in the final score, with $\frac{m}{n}$ in [0,1] as $m \leq n$.

2.3 Context Analyzer

The purpose of *Context Analyzer* is to identify a subset of `DBpedia` nodes representing a context of interest. For instance, if the topics we are interested in are *databases* and *programming languages*, we are interested in the subgraph of `DBpedia` whose nodes are somehow related to *databases* and *programming languages* as well. This subgraph is what we call a **context**. Once we have a context $\mathcal{C}$, given a query represented by a `DBpedia` node u, first we look for the context u belongs to and then we rank nodes in $\mathcal{C}$ with respect to u. In order to identify and compute a context, we use *Graph Explorer* to browse the `DBpedia` graph, starting from an initial meaningful set of resources (**seed nodes**). In this preliminary step a domain expert selects a subset of resources that are representative of the context of interest. The set of seed nodes we selected for the context of databases and programming languages are *PHP*, *Java*, *MySQL*, *Oracle*, *Lisp*, *C#* and *SQLite*.

Since we do not want to explore the whole `DBpedia` graph to compute $\mathcal{C}$, once we reach nodes that we may consider out of the context of interest we need a criterion to stop the exploration. During the exploration we may find some special nodes that are more popular than others, i.e., we may find nodes in the context that are more interconnected to other nodes within $\mathcal{C}$. We call these resources **representative nodes of the context**. Intuitively, given a set of representative nodes of a context $\mathcal{C}$, we may check if a `DBpedia` resource u is within or outside the context of interest evaluating *how relevant* u is with respect to representative nodes of $\mathcal{C}$.

While exploring the graph, *Graph Explorer* populates at each iteration the set $\mathcal{R}$ with nodes representing resources reached starting from the initial seed nodes. Each node contains also information regarding how many times it has been reached during the exploration. The number of hits for a node is incremented every time the corresponding URI is found (see Algorithm 4 in the Appendix). This value is interpreted as "*how popular/important the node is*" within $\mathcal{R}$.

In DBpedia there are a special kind of resources called **categories**[10]. Since in Wikipedia they are used to classify and cluster sets of documents, in DBpedia they classify sets of resources. They might be seen as abstract concepts describing and clustering sets of resources. As a matter of fact, to stress this relation, every DBpedia category is also a `rdf:type skos:Concept`. Moreover, since DBpedia categories have their own labels we may think at these labels as names for clusters of resources. *Context Analyzer* uses these categories in order to find **representative nodes**. In other words, the representative nodes are the *most popular* DBpedia categories in $\mathcal{R}$.

Hence, for each new resource found during the exploration, in order to evaluate if it is within or outside the context, we compare it with the most popular DBpedia categories in $\mathcal{R}$. If the score is greater than a given threshold, we consider the new resource within the context. The value $THRESHOLD$ is set manually. After some tests, for the context of programming languages and database systems, we noticed that a good value for the context we analyzed is $THRESHOLD = 4.0$. Indeed, we noticed that many non-relevant resources were considered as in context if the threshold was lower. On the contrary, a greater value of the threshold was too strict and blocked many relevant resources.

2.4 Storage

For each pair of resources, we store the results computed by *Ranker*. We also keep the information on how many times a resource has been reached during the graph exploration. For each resource belonging to the extracted context, the *Storage* module stores the results returned by *Ranker*. For each resource *root* we store information related to it: $\langle root, hits, ranked, in\_context \rangle$, plus ranking results for each of its discovered nodes u_i: $\langle root, u_i, wikipedia, abstract, google, yahoo, bing, delicious \rangle$.

3 Not Only Tag

In this section we describe a concrete system that exploits the algorithms proposed in Section 2 in order to suggest semantically related tags.

We borrow directly from DBpedia the use case to "*classify documents, annotate them and exploit social bookmarking phenomena*"[11]. Terms from DBpedia can be used to annotate Web content. Following this idea, our system wants to offer a fully-semantic way of social tagging. Figure 3 shows a screenshot of the prototype of the system, *Not Only Tag*, available at `http://sisinflab.poliba.it/not-only-tag`.

The usage is very simple. The users starts by typing some characters (let us say "*Drup*") in the text input area (marked as *(1)* in Figure 3) and the system returns a list of DBpedia resources whose labels or abstracts contain

[10] `http://en.wikipedia.org/wiki/Help:Category`
[11] `http://wiki.dbpedia.org/UseCases#h19-5`

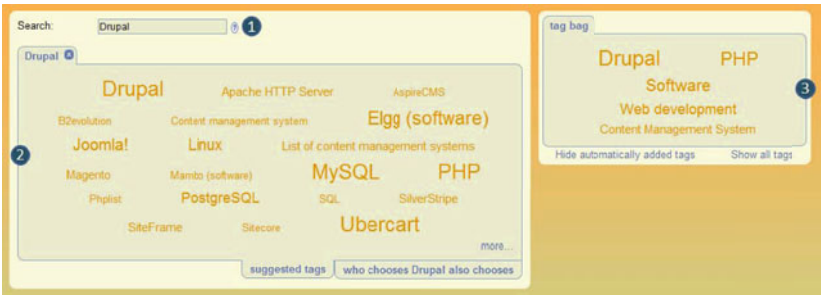

Fig. 3. Screenshot of *Not Only Tag* system

the typed string. Then the user may select one of the suggested items. Let us suppose that the choice is the tag *Drupal*. Then, the system populates a tag cloud (as shown by *(2)* in Figure 3), where the size of the tags reflects their relative relevance with respect to the chosen tag (*Drupal* in this case). The biggest tags are *Ubercart*, *PHP*, *MySQL*, *Elgg* and *Joomla!*. When the user clicks on a whatever tag, the corresponding cloud is created. Thanks to this feature the user can efficiently navigate the `DBpedia` subgraph just like she usually does when jumping from a web page to another one. The user can also drag a tag and drop it in her tag bag area (indicated by *(3)* in Figure 3) or just click on the plus icon next to each tag. Once the user selects a tag, the system enriches this area by populating it with concepts related to the dropped tag. For example, in the case of *Drupal*, its most similar concepts are *PHP*, *Software*, *Web Development*, *Content Management System* and so on. These ancestors correspond to Wikipedia Categories. As seen in Section 2.3 it is possible to discover them because they are the subject of a triple which has `rdf:type` as property and `skos:Concept` as object. Moreover `skos:broader` property links Categories with each other (specifically a subcategory to its category), whereas `skos:subject` relates a resource to a Category. By means of a recursive `SPARQL` query, filtered by the above mentioned properties, it is possible to check if a node is parent of another one.

4 Evaluation

In the experimental evaluation we compared our *DBpediaRanker* algorithm with other four different algorithms; some of them are just a variation of our algorithm but lack of some key features.

Algo2 is equivalent to our algorithm, but it does not take into account textual and link analysis in `DBpedia`.

Algo3 is equivalent to our algorithm, but it does not take into account external information sources, i.e., information coming from search engines and social bookmarking systems.

Algo4, differently from our algorithm, does not exploit textual and link analysis. Moreover, when it queries external information sources, instead of Formula (1), it uses the *co-occurrence* formula: $\frac{p_{u_1,u_2}}{p_{u_1}+p_{u_2}-p_{u_1,u_2}}$.

Algo5 is equivalent to *Algo4*, but it uses the *similarity distance* formula [3] instead of the co-occurrence one.

We did not choose to use either the co-occurrence formula or the similarity distance with *DBpediaRanker* since they do not work well when one of the two resources is extremely more popular than the other, while formula (1) allows to catch this situation.

In order to assess the quality of our proposal we conducted a study where we asked participants to rate the results returned by each algorithm. For each query, we presented five different rankings, each one corresponding to one of the ranking methods. The result lists consisted of the top ten results returned by the respective method. In Figure 4, results for the query *Drupal* are depicted. Looking at all the results obtained with our approach (column 3), we notice that they are really tightly in topic with *Drupal*. For example, if we focus on the first three results, we have *Ubercart*, that is the popular e-commerce module for *Drupal*, *PHP* which is the programming language used in *Drupal*, and *MySQL* the most used DBMS in combinance with *Drupal*. The other results are still very relevant, we have for example *Elgg* and *Joomla!*, that are the major concurrents of *Drupal*, and *Linux* which is the common platform used when developing with *Drupal*.

We point out that even if we use external information sources to perform substantially a textual search (for example checking that the word *Drupal* and the word *Ubercart* appear more often in the same Web pages with respect to the pair *Drupal* and *PHP*), this does not mean that we are discarding semantics in our search and that we are performing just a keyword-based search, as the inputs for the text-based search come from a semantic source. This is more evident if we consider the best results our system returns if the query is *PHP*. In fact, in this case no node having the word *PHP* in the label appears in the first results. On the contrary, the first results are *Zend Framework* and *Zend Engine*, that are respectively the most used web application framework when coding in PHP and the heart of PHP core. *PHP-GTK* is one of the first resources that contains the word *PHP* in its label and is ranked only after the previous ones.

During the evaluation phase, the volunteers were asked to rate the different ranking algorithms from 1 to 5 (as shown in Figure 4), according to which list they deemed represent the best results for each query. The order in which the different algorithms were presented varied for each query: e.g., in Figure 4 the results for *DBpediaRanker* algorithm appear in the third column, a new query would show the results for the same algorithm in a whatever column between the first and the last. This has been decided in order to prevent users to being influenced by previous results.

The area covered by this test was the *ICT* one and in particular *programming languages* and *databases*.

Fig. 4. Screenshot of the evaluation system. The five columns show the results for, respectively, *Algo3*, *Algo4*, *DBpediaRanker*, *Algo2* and *Algo5*.

The test was performed by 50 volunteers during a period of two weeks, the data collected are available at `http://sisinflab.poliba.it/evaluation/data`. The users were Computer Science Engineering master students (last year), Ph.D. students and researchers belonging to the ICT scientific community. For this reason, the testers can be considered IT domain experts. During the testing period we collected 244 votes. It means that each user voted on average about 5 times. The system is still available at the website `http://sisinflab.poliba.it/evaluation`. The user can search for a keyword in the ICT domain by typing it in the text field, or she may directly select a keyword from a list below the text field that changes each time the page is refreshed. While typing the resource to be searched for, the system suggests a list of concepts obtained from `DBpedia`. This list is populated by querying the `DBpedia` URI lookup web service[12].

If the service does not return any result, it means that the typed characters do not have any corresponding resource in `DBpedia`, so the user can not vote on something that is not in the `DBpedia` graph. It may happen that after having chosen a valid keyword (i.e., an existing resource in `DBpedia`) from the suggestion list, the system says that there are no results for the selected keyword. This happens because we used the context analyser (see Section 2.3) to limit the exploration of the `RDF` graph to nodes belonging to *programming languages* and *databases* domain, while the URI lookup web service queries the whole `DBpedia`. In all other cases the user will see a screenshot similar to the one depicted in Figure 4. Hovering the mouse on a cell of a column, the cells in other columns having the same label will be highlighted. This allows to see immediately in which positions the same labels are in the five columns. Finally the user can start to rate the results of the five algorithms, according to the following scale: (i) one star: *very poor*; (ii) two stars: *not that bad*; (iii) three stars: *average*; (iv) four stars: *good*; (v) five stars: *perfect*. The user has to rate each algorithm before sending her vote to the server. Once rated the current resource, the user may vote for a new resource if she wants. For each voting we collected the time elapsed to

[12] `http://lookup.dbpedia.org/api/search.asmx`

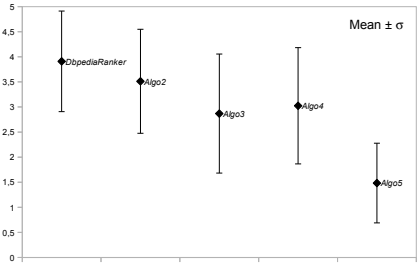

Fig. 5. Average ranks

rate the five algorithms: on the average it took about 1 minute and 40 seconds ($\sigma = 96.03$ s). The most voted resources were `C++`, `MySQL` and `Javascript` with 10 votings.

In Figure 5 we plotted the mean of the votes assigned to each method. Error bars represent standard deviation. *DBpediaRanker* has a mean of 3.91 ($\sigma = 1.0$). It means that, on the average, users rated it as *Good*. Examining its standard deviation, we see that the values are within the range of $\sim 3 \div 5$. In order to determine if the differences between our method and the others are statistically significant we use the Wilcoxon test [13] with $p < .001$. From the Wilcoxon test we can conclude that not only our algorithm performed always better than the others, but also that the (positive) differences between our ranking and the others are statistically significant. Indeed, the *z-ratio* obtained by comparing *DBpediaRanker* algorithm with *Algo2*, *Algo3*, *Algo4* and *Algo5* is respectively 4.93, 8.71, 7.66, 12.89, (with $p < 0.0001$). By comparing these values with the critical value of z[13], we can reject the null hypothesis (correlated rankings), and say that the differences between our algorithm and the others are statistically significant.

5 Related Work

Nowadays, a lot of websites expose their data as `RDF` documents; just to cite a few: the *DBPL* database, *RDF book mashup*, *DBtune*, *MusicBrainz*[14]. `SPARQL` is the *de-facto* standard to query `RDF` datasets. If the considered dataset has a huge dimension, `SPARQL` will return as result of the query a long list of *not-ranked* resources. It would be very useful to have some metrics able to define the relevance of nodes in the `RDF` graph, in order to give back to the user a *ranked* list of results, ranked w.r.t. the user's query. In order to overcome this limit several `PageRank`-like [11] ranking algorithms have been proposed [4,7,9,6]. They seem, in principle, to be good candidates to rank resources in a `RDF` knowledge base. Yet, there are some considerable differences, that cannot be disregard, between ranking web documents and ranking resources to which some semantics

[13] `http://faculty.vassar.edu/lowry/ch12a.html`

[14] `http://www.informatik.uni-trier.de/~ley/db/`, `http://www4.wiwiss.fu-berlin.de/bizer/bookmashup/`, `http://dbtune.org/`, `http://musicbrainz.org/`

is attached. Indeed, the only thing considered by the PageRank algorithm is the origin of the links, as all links between documents have the same relevance, they are just hyperlinks. For RDF resources this assumption is no more true: in a RDF graph there are several types of links, each one with different relevance and different semantics, therefore, differently from the previous case, a RDF graph is not just a graph, but a directed graph with labels on each edge. Moreover a RDF resource can have different origins and can be part of several different contexts and this information has be exploited in some way in the ranking process. *Swoogle* [4] is a semantic web search engine and a metadata search provider, which uses the *OntologyRank* algorithm, inspired by the PageRank algorithm. Differently from *Swoogle*, that ranks RDF documents which *refer* to the query, our main task is to rank RDF resources *similar* to the query. Nonetheless, we borrowed from Swoogle the idea of browsing only a predefined subset of the semantic links. Similarly to our approach also the *ReConRank* [7] algorithm explores just a specific subgraph: when a user performs a query the result is a topical subgraph, which contains all resources related to keywords specified by the user himself. In the subgraph it is possible to include only the nodes *directly* linked to the particular root node (the query) as well as specify the number n of desired hops, that is how far we want to go from the root node. The *ReConRank* algorithm uses a PageRank-like algorithm to compute the relevance of resources, called *ResourceRank*. However, like our approach, the *ReConRank* algorithm tries to take into account not only the relevance of resources, but also the "context" of a certain resource, applying the *ContextRank* algorithm [7]. Our approach differs from [7] due to the semantic richness of the DBpedia graph (in terms of number of links) the full topical graph for each resource would contain a huge number of resources. This is the reason why we only explore the links skos:subject and skos:broader. Hart et al. [6] exploit the notion of naming authority, introduced by [9], to rank data coming from different sources. To this aim they use an algorithm similar to PageRank, adapted to structured information such as the one contained in an RDF graph. However, as for PageRank, their ranking measure is absolute, i.e. it does not depend on the particular query. In our case, we are not interested in an absolute ranking and we do not take into account naming authority because we are referring to DBpedia: the naming authority approach as considered in [6] loses its meaning in the case of a single huge source such as DBpedia. Mukherjea et al. in [10] presented a system to rank RDF resources inspired by [9]. As in the classical PageRank approach the relevance of a resource is decreased when there are a lot of outcoming links from that, nevertheless such an assumption seems not to be right in this case, as if an RDF resource has a lot of outcoming links the relevance of such a resource should be increased not decreased. In our approach, in order to compute if a resource is within or outside the context, we consider as *authority* URIs the most popular DBpedia categories. Based on this observation, URIs within the context can be interpreted as *hub* URIs. *TripleRank* [5], by applying a decomposition of a 3-dimensional tensor that represents an RDF graph, extends the paradigm of two-dimensional graph representation, introduced by HITS, to obtain information on the resources and

predicates of the analyzed graph. In the pre-processing phase they prune dominant predicates, such as `dbpprop:wikilink`, which, instead, have a fundamental role as shown in the experimental evaluation. Moreover in [5] they consider only objects of triples, while we look at both directions of statements. Finally, as for all the HITS-based algorithms, the ranking is just based on the graph structure. On the contrary we also use external information sources. *Sindice* [12], differently from the approaches already presented, does not provide a ranking based on any lexicographic or graph-based information. It ranks resources retrieved by `SPARQL` queries exploiting external ranking services (as Google popularity) and information related to hostnames, relevant statements, dimension of information sources. Differently from our approach, the main task of Sindice is to return `RDF` triples (data) related to a given query. Kasneci et al. [8] present a semantic search engine *NAGA*. It extracts information from several sources on the web and, then, finds relationships between the extracted entities. The system answers to queries about relationships already collected in it, which at the moment of the writing are around one hundred. Differently from our system, in order to query NAGA the user has to know all the relations that can possibly link two entities and has to learn a specific query language, other than know the exact name of the label she is looking for; while we do not require any technical knowledge to our users, just the ability to use tags. We do not collect information from the entire Web, but we rely on the `Linked Data` cloud, and in particular on `DBpedia` at the present moment.

6 Conclusion and Future Work

Motivated by the need of of annotating and retrieving software components in shared repositories, in this paper we presented a novel approach to rank `RDF` resources within the `DBpedia` dataset. The notion of context is introduced to reduce the search space and improve the search results. Semantic resources within a context are ranked according to the query exploiting the semantic structure of the `DBpedia` graph as well as looking for similarity information in web search engines and social tagging systems, and textual and link analysis. The approach has been implemented in a system for semantic tagging recommendation. Experimental results supported by extensive users evaluation show the validity of the approach. Currently, we are mainly investigating how to extract more fine grained contexts and how to enrich the context extracting not only relevant resources but also relevant properties. Moreover we are developing a wrapper for `ProgrammableWeb` using our tagging system as a backend for the annotation process. The aim is to facilitate the tagging process and the subsequently recommendation of software components in the retrieval phase.

Acknowledgment

We are very grateful to Joseph Wakeling for fruitful discussion and to the anonymous users who participated in the evaluation of the system. This research has been supported by Apulia Strategic projects PS_092, PS_121, PS_025.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
2. Bizer, C., et al.: Dbpedia - a crystallization point for the web of data. In: Web Semantics: Science, Services and Agents on the World Wide Web (July 2009)
3. Cilibrasi, R., Vitányi, P.: The Google Similarity Distance. IEEE Transactions on Knowledge and Data Engineering 19(3), 370–383 (2007)
4. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, S.R., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. In: CIKM '04, pp. 652–659 (2004)
5. Franz, T., Schultz, A., Sizov, S., Staab, S.: TripleRank: Ranking semantic web data by tensor decomposition. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 213–228. Springer, Heidelberg (2009)
6. Harth, A., Kinsella, S., Decker, S.: Using naming authority to rank data and ontologies for web search. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 277–292. Springer, Heidelberg (2009)
7. Hogan, A., Harth, A., Decker, S.: ReConRank: A Scalable Ranking Method for Semantic Web Data with Context (2006)
8. Kasneci, G., Suchanek, F.M., Ifrim, G., Ramanath, M., Weikum, G.: Naga: Searching and ranking knowledge. In: ICDE 2008 (2008)
9. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. In: Proc. of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (1998)
10. Mukherjea, S., Bamba, B., Kankar, P.: Information Retrieval and Knowledge Discovery utilizing a BioMedical Patent Semantic Web. IEEE Trans. Knowl. Data Eng. 17(8), 1099–1110 (2005)
11. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Technical report (1998)
12. Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the Open Linked Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 552–565. Springer, Heidelberg (2007)
13. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin 1(6), 80–83 (1945)

Appendix: Algorithms

Algorithm 1. *DBpediaRanker*

Input: a set $\mathcal{S} = \{u_i\}$ of seed nodes
Output: the context

$\mathcal{R} = \emptyset$;
/* For each seed, we create the corresponding node. We impose each seed to be within the context. */
foreach $u_i \in \mathcal{S}$ **do**
  $create\_new\_node(r_i)$;
  $r_i.URI = u_i$;
  $r_i.hits = 1$;
  $r_i.ranked = false$;
  $r_i.in\_context = true$;
  $\mathcal{R} = \mathcal{R} \cup \{r_i\}$;
end
$finished = false$;
while $finished == false$ **do**
  /* We expand only the DBpedia nodes whose corresponding URI is evaluated to be within the context. */
  foreach $r_i \in \mathcal{R}$ *such that both* $(r_i.in\_context == true)$ *and* $(r_i.ranked == false)$ **do**
    $explore(r_i.URI, r_i.URI, MAX\_DEPTH)$;
  end
  $finished = true$;
  /* After we updated $\mathcal{R}$ expanding nodes whose URI is within the context, we might have new representative nodes of the context. Hence, we check if nodes previously considered outside of the context can be reconsidered as part of it. */
  foreach $r_i \in \mathcal{R}$ *such that* $(r_i.in\_context == false)$ **do**
    if $is\_in\_context(r_i.URI)$ **then**
      $r_i.in\_content = true$;
      $finished = false$;
    end
  end
end

Algorithm 2. *explore*$(root, uri, depth)$. The main function implemented in *Graph Explorer*.

Input: a URI *root*; one of *root*'s neighbour URIs *uri*; *depth*: number of hops before the search stops

`/* We perform a depth-first search starting from` *root* `up to a depth of` MAX_DEPTH. `*/`

if $depth < MAX\_DEPTH$ **then**
  if *there exists* $r_i \in \mathcal{R}$ *such that* $r_i.URI == uri$ **then**
    `/* If the resource` *uri* `was reached in a previous recursive step we update its popularity. Moreover, if` *uri* `is evaluated to be within the context we compute how similar` *uri* `and` *root* `are. */`
    $r_i.hits = r_i.hits + 1$;
    if $is\_in\_context(uri)$ **then**
      $sim = similarity(root, uri)$;
    end
  else
    `/* If the resource` *uri* `was not reached in a previous recursive step we create the corresponding node. Moreover, if` *uri* `is evaluated to be within the context we compute how similar` *uri* `and` *root* `are, otherwise we mark` *uri* `as being outside of the context. */`
    $create\_new\_node(\overline{r_i})$;
    $\overline{r_i}.URI = uri$;
    $\overline{r_i}.hits = 1$;
    $\overline{r_i}.ranked = false$;
    if $is\_in\_context(uri)$ **then**
      $sim = similarity(root, uri)$;
      $\overline{r_i}.in\_context = true$;
    else
      $\overline{r_i}.in\_context = false$;
    end
  end
end
`/* If we are not at` MAX_DEPTH `depth w.r.t.` *root*`, we create the set of all the resources reachable from` *uri* `via skos:subject and skos:broader. */`
if $depth > 0$ **then**
  $\mathcal{N} = explode(uri)$;
end
`/* We recursively analyze the resources reached in the previous step. */`
foreach $n_i \in \mathcal{N}$ **do**
  $explore(root, n_i, depth - 1)$;
end
save $\langle root, uri, sim \rangle$;

Algorithm 3. *similarity*(u_1, u_2). The main function implemented in *Ranker*.

Input: two `DBpedia` URIs
Output: a value representing their similarity

$wikipedia = wikiS(u_1, u_2)$;
$abstract = abstractS(u_1, u_2)$;
$google = engineS(u_1, u_2, google)$;
$yahoo = engineS(u_1, u_2, yahoo)$;
$bing = engineS(u_1, u_2, bing)$;
$delicious = socialS(u_1, u_2, delicious)$;
return $wikipedia + abstract + google + yahoo + bing + delicious$;

Algorithm 4. *is_in_context*$(uri, \mathcal{R})$. The main function implemented in *Context Analyzer*.

Input: a DBpedia URI *uri*
Output: *true* if *uri* is considered part of the context, $false$ otherwise

```
1  cont = 0;
2  r = 0;
3  foreach node r ∈ R do
       /*
          We consider the most popular DBpedia categories reached during the exploration
          as the representative URIs of the context.
                                                                                    */
4      if r.URI is one of the ten most popular DBpedia categories reached so far during the
       search then
5          s = s + similarity(uri, r.URI)
6      end
       /*
          If the similarity value computed between uri and the representative URIs of the
          context is greater than a threshold we consider uri as part of the context.
                                                                                    */
7      if s ≥ THRESHOLD then
8          return true
9      end
10 end
11 return false
```