Semantic tag cloud generation via 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. Many current recommender systems exploit textual annotations (tags) provided by users to retrieve and suggest online contents. The text-based recommendation provided by these systems could be enhanced (i) using unambiguous identifiers representative of tags and (ii) exploiting semantic relations among tags which are impossible to be discovered by traditional textual analysis. In this paper we concentrate on annotation and retrieval of web content, exploiting semantic tagging with DBpedia. We use semantic information stored in the DBpedia dataset and propose a new hybrid ranking system to rank keywords and to expand queries formulated by the user. Inputs of our ranking system are (i) the DBpedia dataset; (ii) 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.

Key words: content-based recommendation, RDF ranking, DBpedia

1 Introduction

Many content-based recommender systems exploit textual annotation (e.g., tags) to recommend content to web users. Moreover, in the current Web 2.0 we also see e-commerce sites giving the possibility to users to tag content/products they want to sell/buy, in order to make them easily retrievable by other users willing to buy/sell that content or products[1]. However, the limits of pure textual-based recommender systems are well know: as semantic relations among keywords are not taken into account, they cannot recognize different keywords with the same meaning (synonymy), as well as the fact that a single word can have different meanings (polysemy). This is the main reason why proposed content is sometimes not in topic with what the user is looking for. Pure textual approaches do not allow to face problems such as synonymy, polysemy, homonymy, context analysis, nor to discover particular relations as hyponymy and hyperonymy[2]. Several semantic-based systems exploits ontological information [8, 5, 4] to overcome the above mentioned issues. Unfortuantely, the main problem of such approaches is that it is

[1] Just to cite an example, Amazon allows users to tag products to improve the search and recommendation process (http://www.amazon.com/gp/tagging/cloud/).
[2] www.wikipedia.org/wiki/{Synonym|Polysemy|Homonym|Hyponymy}

very laborious to maintain an ontology regularly updated. Projects like DBpedia[3] may solve the issue of having a semantic source of information regularly updated and which covers a wide range of fields, being based on Wikipedia. Indeed, DBpedia is a community effort to extract structured information from Wikipedia and to make this information available on the Web as a RDF dataset, allowing to pose sophisticated SPARQL queries to Wikipedia. Terms from DBpedia can be used to annotate and represents web contents. Compared to other subject hierarchies and taxonomies, DBpedia has the advantage that each term/resource is endowed with a rich description including abstracts in more than 90 languages. Another advantage, compared to static hierarchies, is that DBpedia evolves as Wikipedia changes. Moreover, each concept in DBpedia is referred by its own URI. This allows to precisely get a resource with no ambiguity. For example, the American corporation *Google Inc.* headquartered in California is referred to as the resource identified by the URI `http://dbpedia.org/resource/Google`, whereas the American comic strip *Barney Google* created in 1919 by Billy DeBeck is referred to as the URI `http://dbpedia.org/resource/Barney_Google_and_Snuffy_Smith`.

The main idea behind our approach is the following: keywords can be mapped to corresponding DBpedia resources. After this mapping, we are able to associate a well defined semantics to keywords and we can enrich the "meaning" of the keywords by exploiting the ontological nature of DBpedia. Main contributions of this work are:

- A tool for the semantic annotation of web resources, useful in both the tagging phase and in the retrieval one (see Section 2).
- A novel *hybrid* approach to rank resources on DBpedia w.r.t. a given keyword. Our system combines the advantages of a *semantic-based* approach (relying on a RDF graph) with the advantages 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, 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 neighborhood 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 ranking list of resources. For this reason we compute a weight representing a similarity relation between resources, instead of a weight for the single resource, as in PageRank-style algorithms.
- 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 our motivating scenario and present a first implementation of the whole system, which is detailed in Section 3. Then, in Section 4 we show and discuss the results of experimental evaluation. In Section 5 we discuss related work with respect to our approach. Conclusion and future work close the paper.

[3] `http://dbpedia.org/`

2 Not Only Tag: a tool for tag cloud generation

In this section we describe a semantic social tagging system *Not Only Tag – NOT* (available at `http://sisinflab.poliba.it/not-only-tag`, see Figure 1)) that can be used to recommend similar tags to users in the annotation and retrieval process of web resources.

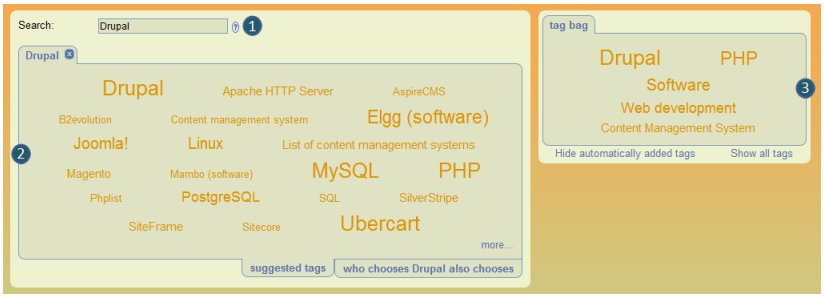

Fig. 1. Screenshot of *Not Only Tag* system.

The interaction with the system is very simple and intuitive. Let us suppose the user wants to annotate a software component. The user starts typing some characters (let us say "*Drup*") in the text input area (marked as *(1)* in Figure 1) and the system suggests a list of `DBpedia` URIs whose labels or abstracts contain the typed string. Then the user may select one of the suggested items. We stress here that the user does not suggest just a keyword but a `DBpedia` resource identified by a unique URI. Let us suppose that the choice is the tag *Drupal*, which corresponds to the URI `dbpres:Drupal`.

The system populates a tag cloud (as shown in Figure 1 *(2)*), where the size of the tags reflects their relative **relevance** with respect to *Drupal* in this case (how the relevance is determined is explained in Section 3). We may see that the biggest tags are *Ubercart*, *PHP*, *MySQL*, *Elgg* and *Joomla!*. If the user goes with the mouse pointer over a tag, the abstract of the corresponding `DBpedia` resource appears in a tooltip. This is useful because it allows for a better understanding of the meaning of that tag. When the user clicks on a tag, the corresponding cloud is created in a new tab. Thanks to this feature the user can also navigate the `DBpedia` subgraph in an intuitive way.

The user can collects suggested tags she consider relevant for her campaign by a drag and drop operation of the tag in her tag-bag area (indicated by *(3)* in Figure 1). Once the user selects a tag, the system automatically enriches this area with concepts related to the dropped tag. For example, in the case of *Drupal*, its most related concepts are *PHP*, *Software*, *Web Development*, *Content Management System* and so on. These new keywords represent the corresponding Wikipedia Categories showed in the Wikipedia page of *Drupal*. Also the tags appearing in the personal tag bag area are sized according to their relevance. Thanks to the `RDF` nature of `DBpedia`, they can be easily computed via nested `SPARQL` queries. In `DBpedia`, for each URI representing Wikipedia category there is a `RDF` triple

having the URI as subject, `rdf:type` as property and `skos:Concept` as object. For a further deeper expansion of (semantic) keywords in the tag bag, we also exploit the `skos:broader` and `skos:subject` properties within `DBpedia`. These two properties are used to represent an ontological taxonomy among Wikipedia categories. In particular, `skos:broader` links a category (subject) to its super-category while `skos:subject` relates a resource to its corresponding Wikipedia category. Finally, the `SPARQL` query used to compute the expanded cloud related to a given resource is recursively repeated for all the related categories.

3 An hybrid algorithm to rank DBpedia resources

In this section we describe our hybrid ranking algorithm *DBpediaRanker*[4], used to rank resources (tags) in `DBpedia` w.r.t. a given keyword. This algorithm computes the *relevance* of `DBpedia` resources (tags) w.r.t. a given keyword and so it allows to determine the *size* of the words in the tag cloud of *NOT* (see Section 2). In a nutshell, *DBpediaRanker* 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 graph browsing, and the consequent ranking of resources, is performed *offline* and, at the end, the result is a weighted graph where nodes are `DBpedia` resources and weights represent the similarity value between any pair of nodes. The graph so obtained will then be used at *runtime*:

- in the *annotation phase*, to suggest *similar* tags to users annotating e.g. their software components;
- in the *retrieval phase*, to display components annotated with tags semantically related to the ones used in the query.

The similarity value between any pair of resources in the `DBpedia` graph is computed querying *external information sources* (search engines and social bookmarking systems) and exploits *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 3.2). Moreover, we look at, respectively, **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 `DBpedia` resource name appears in the abstract of another `DBpedia` 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 link to the Wikipedia page of resource b, and vice versa. In the following we will present in details all the components of our system, whose architecture is sketched in Figure 2.

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

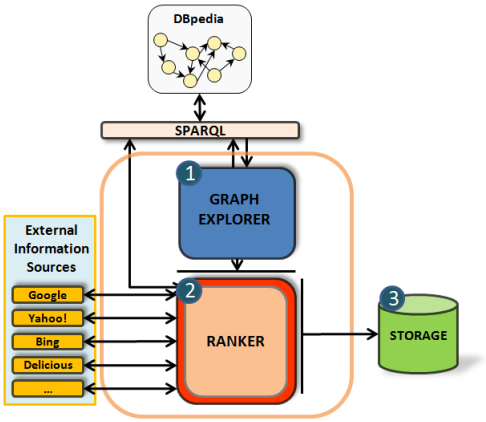

Fig. 2. The ranking system *DBpediaRanker*.

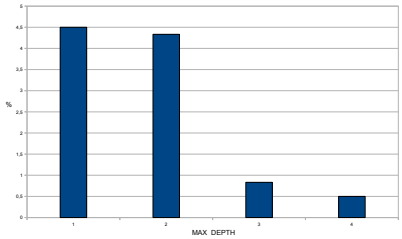

Fig. 3. Evaluation for MAX_DEPTH. It represents the average percentage (y axis) of the top-10 resources related to 100 seeds within a distance of 1 to 4 hops (x axis).

3.1 Graph Explorer

This module queries `DBpedia` via its `SPARQL` endpoint. Given a `DBpedia` URI [5], the explorer looks for other URIs 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` properties `skos:subject` and `skos:broader` Indeed, these two properties are not specific of a particular context and are very popular in the `DBpedia` dataset. Hence, they can be used as a good starting point. 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 root URI, this is explored up to a predefined distance, that can be configured in the initial settings. We found through a series of experiments that setting this distance, that we call MAX_DEPTH, equal to 2 is a good choice. Indeed, resources within two hops are still highly correlated to the root URI, 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 loose 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 3.2). 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 3 the average percentage of top-10 related resources w.r.t. to the distance from a seed (MAX_DEPTH) is shown.

[5] From now on we use the words *URI* and *resource* indistinctly.

3.2 Ranker

Here we describe the ranker, the core component of the whole system. Given any pair of resources in the `DBpedia` graph it determines a similarity value between them; this similarity value is the weight associated to the edge between the two resources.

Given two URIs uri_1 and uri_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.

The aim of this module is to evaluate how strong is a semantic connection between two `DBpedia` resources using information taken from external sources. In our current implementation we consider as external sources both (i) web search engines (Google, Yahoo! and Bing) and (ii) social tagging systems (Delicious), plus (iii) Wikipedia-related information contained in `DBpedia`. Given two `DBpedia` resources uri_1 and uri_2, we verify how many web pages contain (or have been tagged by) the value of the `rdfs:label` associated to uri_1 and uri_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 URIs uri_1 and uri_2 with respect to a given external information source $info\_source$.

$$sim(uri_1, uri_2, info\_source) = \frac{p_{uri_1,uri_2}}{p_{uri_1}} + \frac{p_{uri_1,uri_2}}{p_{uri_2}} \tag{1}$$

Given the information source $info\_source$, p_{uri_1} and p_{uri_2} represent the number of documents containing (or tagged by) the `rdfs:label` associated to uri_1 and uri_2 respectively, while p_{uri_1,uri_2} represents how many documents contain (or have been tagged by) both the label of uri_1 and uri_2. It is easy to see that the formula is symmetric and the returned value is in $[0, 2]$. The number of documents containing both labels has to be always lower or equal to those containing only one of the two labels. Otherwise we set the value $\frac{p_{uri_1,uri_2}}{p_{uri_1}} = 0$.

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 resource URIs uri_1 and uri_2. Hence, if there is a `dbpprop:wikilink` from uri_1 to uri_2 and/or vice versa, we assume a stronger relation between the two resources. We evaluate the strength of the connection as follow:

$$wikiS(uri_1, uri_2) = \begin{cases} 0, \text{ no } \texttt{wikilink} \text{ between } uri_1 \text{ and } uri_2; \\ 1, \texttt{wikilink} \text{ only from } uri_1 \text{ to } uri_2; \\ 1, \texttt{wikilink} \text{ only from } uri_2 \text{ to } uri_1; \\ 2, \texttt{wikilink} \text{ both from } uri_1 \text{ to } uri_2 \text{ and} \\ \quad \text{vice versa}; \end{cases}$$

Furthermore, given two resources uri_1 and uri_2, we check if the `rdfs:label` of uri_1 is contained in the `dbpprop:abstract` of uri_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, $abstractS(uri_1, uri_2) = \frac{m}{n}$, with $\frac{m}{n}$ in [0,1] as $m \leq n$. At the end, the similarity value between uri_1 and uri_2 is computed as the sum of the functions:

$$\begin{aligned} &sim(uri_1, uri_2, google) + sim(uri_1, uri_2, yahoo) + sim(uri_1, uri_2, bing) + \\ &+ sim(uri_1, uri_2, delicious) + wikiS(uri_1, uri_2) + abstractS(uri_1, uri_2) \end{aligned} \tag{2}$$

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 the formula (1), it uses the *co-occurrence* formula: $\frac{p_{uri_1, uri_2}}{p_{uri_1} + p_{uri_2} - p_{uri_1, uri_2}}$

Algo5 is equivalent to *Algo4*, but it uses *similarity distance* [1] instead of co-occurrence.

We did not choose to use either co-occurrence or similarity distance in *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 algorithm we conducted a study where we asked to 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 really are 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*. It is very likely that a user who knows *Drupal*, also knows the languages and technologies our measure returned.

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

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

Drupal and *PHP*), this does not mean that we are discarding semantics in our search and that we are performing just a string comparison. Indeed, we are *not* performing just a keyword-based search: this is still 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. For the same reason the columns do not have the name of the ranking measure.

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

The test was performed by 50 volunteers during a period of two weeks. 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 `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`.

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. It happens because we limited 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`. For the sake of simplicity, in this first experiment we decided not to filter results from the URI lookup web service with the nodes in our context. Occasionally it may happen that a keyword belonging to IT domain gives no results, this could happen because the selected resource has not yet been analyzed by *DBpediaRanker*.

In all other cases the user will see a screenshot similar to the one depicted in Figure 4. Moving the mouse pointer on a cell of a column, the cells in other columns having the same label will be highlighted. This allows the user to better understand how differently algorithms rank the same resource and in which positions the same labels are in the five columns. Clicking on a concept, the corresponding Wikipedia page will open in an iframe. This facilitates the user to obtain more information about the clicked concept.

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 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 value 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 $\sim 3 \div 5$, i.e., the ranks are comprised between *Average* and *Perfect*. In order to determine if the differences between our method and the others are statistically significant we used the Wilcoxon test [13]. From the Wilcoxon test we can conclude that not only our

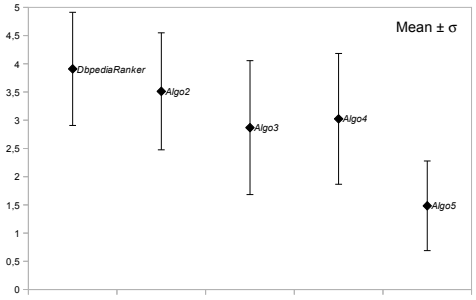

Fig. 5. Average ranks.

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[6], 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*[7]. Therefore, 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 [2, 7, 9, 6]. They seem, in principle, to be good candidates to rank resources in an RDF knowledge base. Yet, there are some considerable differences, that cannot be disregard, between ranking web documents and ranking resources to which some semantics 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 an RDF graph there are several types of links, each one with different relevance and different semantics, therefore, differently from the previous case, an RDF graph is not just a graph, but a directed graph with labels on each edge. Moreover an RDF resource can have different origins and can be part of several different contexts and this information cannot be disregarded, instead it should be exploited in some way in the ranking process. *Swoogle* [2] 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

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

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

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. In order to achieve 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* [3], 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 [3] 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

In this paper we presented a novel system for semantic tag generation and retrieval. We motivated our approach in a scenario of annotation of web resources, showing how exploiting semantic information in `DBpedia` it is possible both (i) to help

users in the process of tag selection, and (ii) to enhance the retrieval process of previously annotated content, displaying the most relevant resources w.r.t. the keywords (tags) specified by the user. We described the components of our system and showed the validity of our approach through experimental results supported by extensive users evaluation. Currently, we are mainly investigating on how to extract more fine grained contexts and how to enrich the context extracting not only relevant resources but also relevant properties.

Acknowledgment

This research has been supported by Apulia Strategic projects PS_092, PS_121, PS_025.

References

1. R. Cilibrasi and P. Vitányi. The Google Similarity Distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.
2. L. Ding, T. Finin, A. Joshi, R. Pan, S. R. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM '04*, pages 652–659, 2004.
3. T. Franz, A. Schultz, S. Sizov, and S. Staab. Triplerank: Ranking semantic web data by tensor decomposition. In *ISWC*, 2009.
4. E. Gabrilovich and S. Markovitch. Harnessing the expertise of 70,000 human editors: Knowledge-based feature generation for text categorization. *J. Mach. Learn. Res.*, 8:2297–2345, 2007.
5. S. C. Gates, W. Teiken, and K.-S. F. Cheng. Taxonomies by the numbers: building high-performance taxonomies. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 568–577. ACM, 2005.
6. A. Harth, S. Kinsella, and S. Decker. Using naming authority to rank data and ontologies for web search. In *International Semantic Web Conference*, 2009.
7. A. Hogan, A. Harth, and S. Decker. ReConRank: A Scalable Ranking Method for Semantic Web Data with Context. 2006.
8. G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *ICDE 2008*, 2008.
9. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, 1998.
10. S. Mukherjea, B. Bamba, and P. Kankar. Information Retrieval and Knowledge Discovery utilizing a BioMedical Patent Semantic Web. *IEEE Trans. Knowl. Data Eng.*, 17(8):1099–1110, 2005.
11. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, 1998.
12. G. Tummarello, R. Delbru, and E. Oren. Sindice.com: Weaving the Open Linked Data. *The Semantic Web*, pages 552–565, 2008.
13. F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.