

Semantic Tags Generation and Retrieval for Online Advertising

Roberto Mirizzi
Politecnico di Bari
Via Orabona, 4
70125 Bari, Italy
mirizzi@deemail.poliba.it

Azzurra Ragone
University of Trento
Via Sommarive, 14
38100 Povo (Trento), Italy
ragone@disi.unitn.it

Tommaso Di Noia,
Eugenio Di Sciascio
Politecnico di Bari
Via Orabona, 4
70125 Bari, Italy
{dinoia, disciascio}@poliba.it

ABSTRACT

One of the main problems in online advertising is to display ads which are relevant and appropriate w.r.t. what the user is looking for. Often search engines fail to reach this goal as they do not consider semantics attached to keywords. In this paper we propose a system that tackles the problem by two different angles: help (i) advertisers to create more efficient ads campaigns and (ii) ads providers to properly match ads content to keywords in search engines. We exploit semantic relations stored in the DBpedia dataset and use an 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.

Categories and Subject Descriptors

H.3.3 [Information Systems]: Information Search and Retrieval

General Terms

Algorithms

1. INTRODUCTION

Computational Advertising is a new field of research, fostered by the rapid growth of advertising on the web. Its aim is to develop new technologies, methodologies and tools which can help current systems to retrieve the best ads belonging to a given context.

Web Advertising dates back to around ten years ago, when short text advertisements began to be inserted among the results of queries to search engines (the so called *Sponsored Search*) and in the content of web pages (a.k.a. *Content*

Match). In the meantime, new methodologies and smart techniques have been developed mainly based on sophisticated statistical analysis on plain and structured text. Nevertheless, these techniques often do not take into account semantic relations among keywords, displaying ads that are sometimes not relevant w.r.t. what the user is looking for or the text of the web page where the ad is placed (e.g. ads about a *zoo* for a page talking about *Tiger Woods*). Moreover, advertisers lose a lot of potentially interested users because they do not exploit all the possible combinations of (semantically related) keywords and phrases that could be used by the users in a query to a search engine. This means that a large slice of clients/customers is usually neglected because of the difficulty required by the creation of a successful marketing campaign.

The next step in computational advertising is finding a new technology able both (i) to improve the relevance/appropriateness of the ads in order to better capture the user's attention and (ii) to ease the process of ad creation allowing the advertiser to establish powerful campaigns in a simplified way.

Currently, the process of ads generation is quite tedious: one of the most boring task is the selection of keywords and bid phrases that activate a given ad. In fact, since its birth, the relevance of an ad with respect to a given query or to the content of a web page has been determined above all by statistical text analysis. The simplicity of this approach has several drawbacks: considering only a lexicographic approach and discarding the semantics of phrases does not allow to face problems such as synonymy, polysemy, homonymy, context analysis, nor to discover particular relations as hyponymy and hyperonymy¹. If we consider only a string-based analysis, we can not *semantically expand* both queries and ads. As a result, if two objects (e.g., an *ad* and a *query*, or an *ad* and a *web page*) use different collections of words to represent similar topics, they will be assigned to different clusters and will not match, although the meaning conveyed by both objects is related.

In the last years, several works [4, 8, 3] have been proposed to overcome the above mentioned problems. Many of them identified a possible solution in the adoption of external sources of information. By using a well structured information source as external domain knowledge (a taxonomy or an ontology), the proposed solutions tend to classify an ad, a query or a web page, in a set of resources belong-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.
Copyright 2010 ACM 978-1-4503-0099-5/10/10 ...\$10.00.

¹ www.wikipedia.org/wiki/{Synonym|Polysemy|Homonym|Hyponymy}

ing to this external knowledge, which are linked to other resources according to precise relationships. This allows to add semantics to objects traditionally analyzed just on a syntactic/textual base. However the drawback of such approaches is that is very laborious to maintain an ontology regularly updated. Moreover it is very difficult to develop an ontology which covers every possible domain (advertising can promote products and services belonging to many different contexts). Projects like Wikipedia, being a multilingual written collaboratively encyclopedia, may solve the issue of coverage and updating. In fact, Wikipedia has more than 90 thousands active contributors working on more than 15 millions articles in more than 270 language². It is a huge source of knowledge, but it is mainly conceived for being used by humans. An agent can not fully exploit easily the semantics of articles and relations among them to discover new information. This is one of the main reasons behind the birth of the DBpedia project (see Section 2.2). DBpedia is a community effort to extract structured information from Wikipedia and to make this information available on the Web as a RDF³ dataset, allowing to pose sophisticated queries to Wikipedia. Terms from DBpedia can be used to annotate and represents ads and web contents.

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. We associate a set of semantically related resources to each keyword mapped to a DBpedia resource. Moreover, we need to associate only new resources belonging to the same context of the original query. Hence, we need procedures able to identify a context and to identify all the DBpedia resources related to the identified context. Furthermore, to expand the query we also need to “rank” in some way DBpedia resources w.r.t. the resources in the query itself, in order to identify the most similar ones.

Main contributions of this work are:

- A system for semantic keyword expansion useful both in the (i) displaying of the *most appropriate* ads in sponsored search auctions and (ii) for suggesting keywords to advertisers preparing their campaigns.
- 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

²<http://en.wikipedia.org/wiki/Wikipedia:About>

³<http://www.w3.org/RDF/>

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 examine the main technologies behind our approach. In Section 3 we introduce our motivating scenario and present a first implementation of the whole system, which is detailed in Section 4. Then, in Section 5 we show and discuss the results of experimental evaluation. In Section 6 we discuss related work with respect to our approach. Conclusion and future work close the paper.

2. BACKGROUND TECHNOLOGIES

2.1 SPARQL

SPARQL⁴ (recursive acronym of *SPARQL Protocol and RDF Query Language*) is a RDF query language, become an official W3C Recommendation in the beginning of 2008. It can be used to perform queries on RDF data sources, whether the data is stored natively as RDF or extracted as RDF through middleware. SPARQL can return results either as result sets or RDF graphs. Usually a SPARQL query is composed by at least a basic graph pattern, that is a set of triple patterns. These patterns are similar to RDF triples, being composed by a *subject*, a *predicate* and an *object*, but here each of the subject, predicate and object may also be a variable. A basic graph pattern matches a subgraph of the whole RDF graph when RDF terms of the subgraph can be replaced with the variables of the query and the result is an RDF graph which is equivalent to the subgraph. The query syntax is for some aspect similar to SQL, but its semantics is different. For example, a simple query to retrieve some information according to some constraints consists of two parts: the **SELECT** clause identifies the variables to appear in the query results, and the **WHERE** clause provides the graph pattern to match against the data graph. The graph pattern can be a single triple with a single variable or more complex, such as a combination of triples with several variables and other smaller patterns.

2.2 Linked Data and DBpedia

The idea behind **Linked Data** [1] is about using the Web to allow linking data and easing the publication of new linked data on the Web. It describes a new method of exposing, connecting and sharing data through dereferenceable URIs on the Web. The goal is to extend the Web by publishing various open datasets as RDF triples on the Web and by setting RDF links between data items from different data sources. According to this aim, URIs are fundamental to identify everything. Using HTTP URIs, things can be referred to and looked up both by people and by agents. **Linked Data** uses RDF to describe things in the world.

DBpedia [2] is one of the main cloud of the **Linked Data** graph. It is the machine-understandable equivalent of Wikipedia project. It is possible to ask queries against DBpedia (through its SPARQL endpoint <http://dbpedia.org/>

⁴<http://www.w3.org/TR/rdf-sparql-query/>

sparql), and link other data sets on the web to DBpedia data. At the present moment DBpedia contains over one billion pieces of information. All this information is stored in RDF triples. DBpedia labels and abstracts of resources are stored multiple languages. The graph is highly connected to other RDF datasets of the Linked Data cloud. Compared to other subject hierarchies and taxonomies, DBpedia has the advantage that each term/resource is endowed with a rich description including abstracts. Another advantage compared to static hierarchies is that DBpedia evolves as Wikipedia changes. Hence, problems such as domain coverage, content freshness, machine-understandability can be addressed more easily when considering DBpedia. 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 *Yahoo!* headquartered in California is referred to as the resource identified by the URI <http://dbpedia.org/resource/Yahoo!>, whereas the legendary *Yahoo* being in Gulliver's Travels is referred to as the URI [http://dbpedia.org/resource/Yahoo_\(Gulliver's_Travels\)](http://dbpedia.org/resource/Yahoo_(Gulliver's_Travels)).

The DBpedia RDF dataset is hosted and published using *OpenLink Virtuoso*⁵. This infrastructure gives access to DBpedia's RDF data through a SPARQL endpoint, with HTTP support for any Web client's standard GETs for HTML or RDF representations of DBpedia resources.

Wikipedia authors try to organize articles by topics, clustering them into Categories⁶. They group together articles on related topics. Most categories can contain several subcategories, i.e., categories that are more specific than their parents. In order to avoid an explosion of articles belonging to a given category, articles are not usually placed in every category which they logically belong to. In many cases they are placed just in the more specific subcategories. For this reason, in Wikipedia it may be necessary to explore in depth to find all the categories an article belongs to. In DBpedia, the relations among categories and articles are elicited through the SKOS vocabulary⁷. In particular `skos:broader` property links a category (that is the subject of a RDF triple) to its super-category, while `skos:subject` relates a resource to its corresponding Wikipedia category. Actually the `skos:subject` property has been deprecated from the SKOS vocabulary, nevertheless it has not yet been replaced by any other property⁸.

3. NOT ONLY TAG: SEMANTIC TAG CLOUD GENERATION

Computational advertising is a wide research topic whose main challenges are: improving ad selection, behavioral targeting, advertiser experience, cost of serving and scalability, forecasting user visits. The approach and system we present here propose a semantic-based approach to cope with two of them: improving ad selection and advertiser experience. A large slice of web advertising consists of textual ads, i.e., short pieces of text included among the results of the search engines (*Sponsored Search*) or among the content of web pages (*Content Match* or *Contextual Advertising*). Here, we

⁵ <http://virtuoso.openlinksw.com/>

⁶ <http://en.wikipedia.org/wiki/Help:Category>

⁷ <http://www.w3.org/2004/02/skos/>

⁸ A natural candidate for the replacement could be the `dc:subject` property, belonging to *Dublin Core* specification, as proposed in <http://www.w3.org/2006/07/SWD/wiki/SkosDesign/Indexing>.

focus on Sponsored Search, where it is crucial that the ads displayed to users are relevant w.r.t. the query and, above all, belong to the same context. Unfortunately, this does not always happen since the selection of ads is mainly based on text-based IR techniques and so semantics of both the query and ads is not taken into account. For this reason, a lot of ads shown today brings little value to the users. Moreover, the selection of ads is determined by an auction process, involving several parameters, including bids on specific keywords. In other words, the advertiser has to carefully choose the keywords (and the bids) that determine when her ads should be activated. Failing the right choice, or neglecting some keywords, causes the advertisement not to be displayed and, consequently, the advertiser to be ignored by a large part of web users.

The above two issues share the same basic problem: given a keyword/tag, there is the need to find semantically related ones. Indeed, in the ad selection process in a sponsored search we need to find ads whose related keywords are semantically related to the query. Similarly, in the keyword selection during the creation of the advertising campaign, the advertiser needs to add new tags whose meaning (semantics) is related to the one of the keywords she originally selected.

The system we describe in the following (*Not Only Tag – NOT*) aims to overcome the above issues by enriching a keyword/tag with semantic information coming from DBpedia. The system makes possible to discover the *meaning* of a query in order to show to the user those ads that are in the same context (and meaning) of the query itself.

Moreover, *NOT* is really useful also to help advertisers preparing their campaigns. If the advertiser is willing to promote a web site on *Luxury cars*, she could be suggested that also *BMW M5* and *Jaguar FX* are suitable keywords for the campaign, with (possibly) a different semantic similarity degree.

We are interested in *contextualizing* and *enriching* the initial set of keywords with the support of a semantic-based recommendation system: the system suggests relevant keywords (extracted from DBpedia) belonging to the same context and domain of an initial set of specified tags. The process is analogous both in the ad selection phase and during the preparation of the campaign: the user query can be disambiguated (thanks to the uniqueness of DBpedia URIs) and then expanded through semantics information contained in DBpedia. Note that, thanks to the uniqueness of DBpedia URIs identifying suggested tags we do not suffer from problems such as synonymy or polysemy as for simple keywords⁹.

A working prototype of the proposed system of *Not Only Tag (NOT)* is available at <http://sisinflab.poliba.it/not-only-tag> (see Figure 1).

Here we show the version of *NOT* used to select the right keywords in the creation of an advertising campaign. The interaction with the system is very simple and intuitive. The user starts to type 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 ab-

⁹ Due to limited computational resources, currently we only consider the portion of DBpedia related to the context of *databases* and *programming languages*. Nevertheless, we point out that our approach is not tied to any particular context, indeed it can be applied to whatsoever domain. We are expanding the explored domains to cover the whole DBpedia.

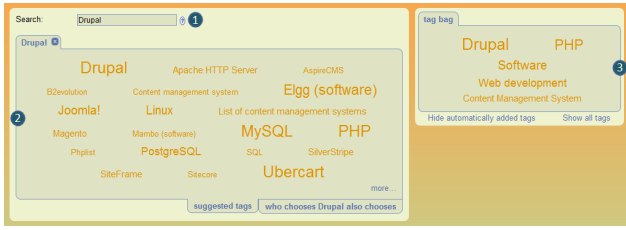


Figure 1: Screenshot of *Not Only Tag* system.

stracts contain the typed string. Then the user may select one of the suggested items. Again, we stress here that the system does not suggest just a keyword but a DBpedia resource identified by a unique URI (e.g., the user can easily distinguish between the resource about *Java (programming language)* and *Java (island)*). Let us suppose that the choice is the tag *Drupal*, which corresponds to the URI <http://dbpedia.org/resource/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. 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 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 collect suggested tags she considers relevant for her campaign by a drag and drop operation of the tags 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*. The tags appearing in the personal tag bag area are sized according to their relevance. Thanks to the RDF nature of DBpedia, categories 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. As an example for the Wikipedia category *Content Management Systems*, in DBpedia we have:

```
dbpres:Category:Content_management_systems rdf:type skos:Concept
```

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 (see Section 2.2):

```
dbpres:Drupal skos:subject dbpres:Category:Content_management_systems
```

```
dbpres:Category:Content_management_systems skos:broader
dbpres:Category:Web_development_software
```

The SPARQL query used to compute the expanded cloud related to a given resource is recursively repeated for all the related categories. For example, for the *Drupal* resource the first query is:

```
SELECT DISTINCT ?hasValue WHERE {
  dbpres:Drupal ?p ?hasValue.
  ?hasValue rdf:type skos:Concept
  FILTER(?p = skos:subject || ?p = skos:broader)
}
```

4. DBPEDIARANKER: RDF RANKING IN DBPEDIA

In this section we will describe our hybrid ranking algorithm *DBpediaRanker*, used to rank resources in DBpedia w.r.t. a given keyword.

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 the nodes are DBpedia resources and the weights represent the similarity value between any pair of nodes. The graph so obtained will then be used at *runtime*: (i) in the *ad creation phase*, to suggest *similar* keywords to advertisers creating their campaign, and (ii) in the *ad selection phase*, to display ads semantically related to the query (keywords) posed to the search engine.

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 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 4.2). Moreover, we look at, respectively, **abstracts** in Wikipedia and **wikilinks**, i.e., links between Wikipedia pages. Specifically, given two nodes uri_1 and uri_2 , we check if the label of node uri_1 is contained in the abstract of node uri_2 , 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 uri_1 has a link to the Wikipedia page of resource uri_2 , and vice versa.

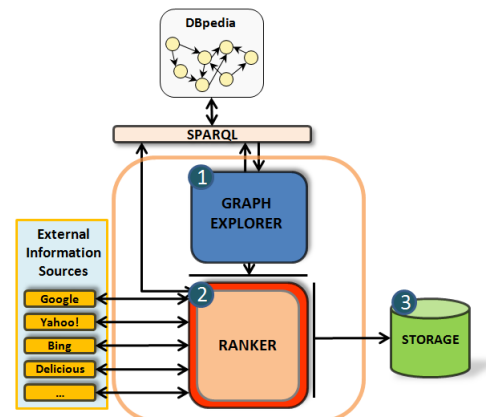


Figure 2: The ranking system *DBpediaRanker*.

The main data structure we use in the approach contains information about DBpedia resources (URIs¹⁰) reached during the exploration. Hence, for each reached resource there is an associated data structure r that is defined as:

$$r = \begin{cases} \text{URI} & : \text{ a DBpedia URI;} \\ \text{hits} & : \text{ how many times the URI has been} \\ & \text{reached during the exploration;} \\ \text{ranked} & : \text{ boolean value representing if the} \\ & \text{URI has already been ranked} \\ & \text{w.r.t. its neighbors;} \\ \text{in_context} & : \text{ boolean value stating if the URI} \\ & \text{has to be considered within the} \\ & \text{context or not;} \end{cases}$$

Now we detail the behavior of each module with respect to Algorithm 1 summarizing the whole ranking procedure. 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. If you want to explore only a domain specific portion of the whole DBpedia graph, context experts have to select the seed nodes (see Section 4.3). The algorithm explores the DBpedia graph using a depth-first approach up to a depth of MAX_DEPTH (see Section 4.1). The function *is_in_context* [line 17 of Algorithm 1] compares each node with representative nodes to decide if it is in the context or not, and so has to be explored further. In the following we will present in details all the components of our system, whose functional architecture is sketched in Figure 2.

Algorithm 1: DBpediaRanker

```

Input: a set  $\mathcal{S} = \{u_i\}$  of seed nodes
1  $\mathcal{R} = \emptyset$ ;
   /* For each seed, we create the corresponding node. We
   impose each seed to be within the context. */
2 foreach  $u_i \in \mathcal{S}$  do
3   create_new_node( $r_i$ );
4    $r_i.URI = u_i$ ;
5    $r_i.hits = 1$ ;
6    $r_i.ranked = false$ ;
7    $r_i.in\_context = true$ ;
8    $\mathcal{R} = \mathcal{R} \cup \{r_i\}$ ;
9 end
10  $finished = false$ ;
11 while  $finished == false$  do
   /* We expand only the DBpedia nodes whose corresponding
   URI is evaluated to be within the context. */
12 foreach  $r_i \in \mathcal{R}$  such that both ( $r_i.in\_context == true$ ) and
   ( $r_i.ranked == false$ ) do
13   | explore( $r_i.URI, r_i.URI, MAX\_DEPTH$ );
14 end
15    $finished = true$ ;
   /* 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. */
16 foreach  $r_i \in \mathcal{R}$  such that ( $r_i.in\_context == false$ ) do
17   | if is_in_context( $r_i.URI$ ) then
18     | |  $r_i.in\_context = true$ ;
19     | |  $finished = false$ ;
20   | end
21 end
22 end

```

4.1 Graph Explorer

This module (see Algorithm 2) queries DBpedia via its SPARQL endpoint. Given a DBpedia URI, the explorer looks

¹⁰ From now on we use the words *URI*, *resource* and *node* indistinctly.

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

```

Input: a URI  $root$ ; one of  $root$ 's neighbor URIs  $uri$ ;  $depth$ :
number of hops before the search stops
1  $sim = 0$ ;
   /* We perform a depth-first search starting from  $root$  up to
   a depth of  $MAX\_DEPTH$ . */
2 if  $depth < MAX\_DEPTH$  then
3   | 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.
   | | */
   | | 4  $r_i.hits = r_i.hits + 1$ ;
   | | 5 if is_in_context( $uri$ ) then
   | | 6 | |  $sim = similarity(root, uri)$ ;
   | | 7 | | end
   | 8 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. */
   | | 9 create_new_node( $\bar{r}_i$ );
   | | 10  $\bar{r}_i.URI = uri$ ;
   | | 11  $\bar{r}_i.hits = 1$ ;
   | | 12  $\bar{r}_i.ranked = false$ ;
   | | 13 if is_in_context( $uri$ ) then
   | | 14 | |  $sim = similarity(root, uri)$ ;
   | | 15 | |  $\bar{r}_i.in\_context = true$ ;
   | | 16 else
   | | 17 | |  $\bar{r}_i.in\_context = false$ ;
   | | 18 | | end
   | 19 end
20 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. */
21 if  $depth > 0$  then
22 |  $\mathcal{N} = expand(uri)$ ;
23 end
   /* We recursively analyze the resources reached in the
   previous step. */
24 foreach  $n_i \in \mathcal{N}$  do
25 | | explore( $root, n_i, depth - 1$ );
26 end
27 save ( $root, uri, sim$ );

```

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

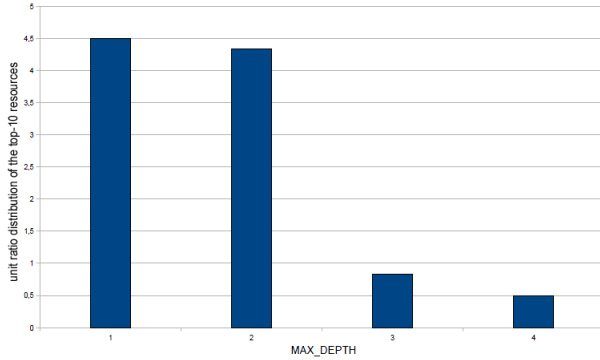


Figure 3: Evaluation for *MAX_DEPTH*. It represents the average percentage (*y* axis) of the top-10 resources related to 100 seeds within a distance of 1 to 4 hops (*x* axis).

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 4.2). 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.

4.2 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 re-

sources 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 $info_source$.

$$sim(uri_1, uri_2, info_source) = \left(\frac{p_{uri_1, uri_2}}{p_{uri_1}} + \frac{p_{uri_1, uri_2}}{p_{uri_2}} \right)_{info_source} \quad (1)$$

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 $[0, 2]$.

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 to 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:

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

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 weighted sum of the functions:

$$\begin{aligned} similarity(uri_1, uri_2) = & w_{google} * sim(uri_1, uri_2, google) + \\ & w_{yahoo} * sim(uri_1, uri_2, yahoo) + w_{bing} * sim(uri_1, uri_2, bing) + \\ & w_{delicious} * sim(uri_1, uri_2, delicious) + w_{wikiS} * wikiS(uri_1, uri_2) + \\ & w_{abstract} * abstractS(uri_1, uri_2) \end{aligned}$$

In *N.O.T.* (see Section 3) we set all the weights $w = 1$.

4.3 Context Analyzer

The purpose of *Context Analyzer* is to identify a subset of DBpedia nodes representing a context of interest. For instance, if the advertising content network is just centered on, e.g., *databases* and *programming languages*, we are interested only in the exploration and ranking of the subgraph of DBpedia whose nodes are somehow related to *databases* and *programming languages* as well. This subgraph is what we call a **context**. The context is represented by the most popular Wikipedia *Categories* reached during the exploration of the graph, i.e., by the *Categories* that are encountered more often and for this reason are more interconnected to other

#	Resource
1.	Programming languages
2.	Object-oriented programming languages
3.	Cross-platform software
4.	Software
5.	Programming language families
6.	Software by operating system
7.	Database management systems
8.	Scripting languages
9.	Free software projects
10.	Computer libraries

Table 1: The ten most popular DBpedia Categories after the analysis of five thousand resources.

nodes. For example, in the selected domain, the ten most popular categories are represented in Table 1.

Once we have a context C , given a query represented by a DBpedia node uri , we check if uri belongs to the context C , ranking uri w.r.t. to all the representative nodes of C , i.e., the top *Categories* (see Algorithm 3). 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 C (i.e., with the representative nodes of C). If the score is greater than a given threshold, we consider the new resource within the context. The value *THRESHOLD* is set manually. In order to find an optimal value for the threshold, we executed several test on the first fifty discovered nodes starting from seven seed nodes (see below). In these tests we considered different values for *THRESHOLD*, in the range [1.0, 10.0], at intervals of 0.5. After manual evaluation we noticed that the best value for the context we analyzed is *THRESHOLD* = 4.0. Indeed, we observed 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.

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** that in this case have to belong to the selected domain and must be selected by domain experts). In order to verify if there is an optimal number of initial seeds, we performed two tests, changing the number of the initial seed nodes. In the first test we set *seed_nodes* = 7, in the second *seed_nodes* = 100. In spite of the small number of seeds of the first test, we were able to discover approximatively the same domain-specific subgraph of DBpedia of the second test, due to highly interconnected nature of the graph.

5. 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

Algorithm 3: *is_in_context(uri)*. 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  $s = 0$ ;
  /* We consider the most popular DBpedia categories reached
   during the exploration as the representative URIs of the
   context  $C$ . */
2  $C = getContext()$ ;
3 foreach node  $r \in C$  do
4    $s = s + similarity(uri, r.URI)$ 
   /* 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. */
5   if  $s \geq THRESHOLD$  then
6     | return true
7   end
8 end
9 return false

```

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* [5] 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. The five algorithms are summarized in Table 2.

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 combination 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 is searching for *Drupal*, probably is also interested in the languages and technologies our measure returned. The results are very interesting in an advertising scenario, since none of these keywords could be retrieved just with text analysis.

Method	Description
<i>DBpediaRanker</i>	External Information Sources Text and Links Analysis
<i>Algo2</i>	External Information Sources
<i>Algo3</i>	Text and Links Analysis
<i>Algo4</i>	External Information Sources Ranking with co-occurrence formula
<i>Algo5</i>	External Information Sources Ranking with Similarity Distance formula

Table 2: Ranking methods evaluated.

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 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. If we think at *keyword suggestion tools*¹¹ used for the expansion of keywords, usually they suggest words (or groups of words) that are *syntactically* similar to the original one. On the contrary, being our system semantic-based, we are able to suggest items that are *semantically* related to the original.

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 are not labeled with 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*. 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

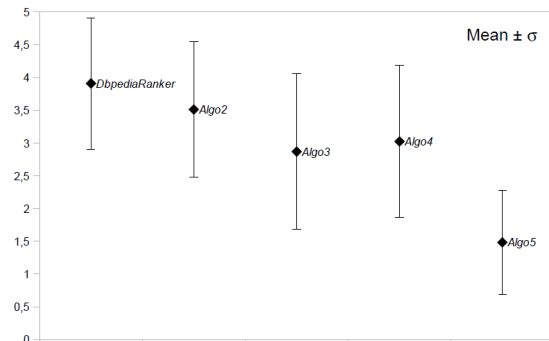


Figure 5: Average ranks.

see a screenshot similar to the one depicted in Figure 4. Moving the mouse pointer on a cell of a column, the cells in the 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. 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 (the full list of collected votes is available at <http://sisinflab.poliba.it/evaluation/data>). *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 [16]. 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 ¹², we can reject the null hypothesis (correlated rankings), and say that the differences between our algorithm and the others are statistically significant.

6. 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*¹³. SPARQL is the *de-facto*

¹¹e.g., <http://www.google.com/sktool/>.

¹²<http://faculty.vassar.edu/lowry/ch12a.html>

¹³<http://www.informatik.uni-trier.de/~ley/db/>,

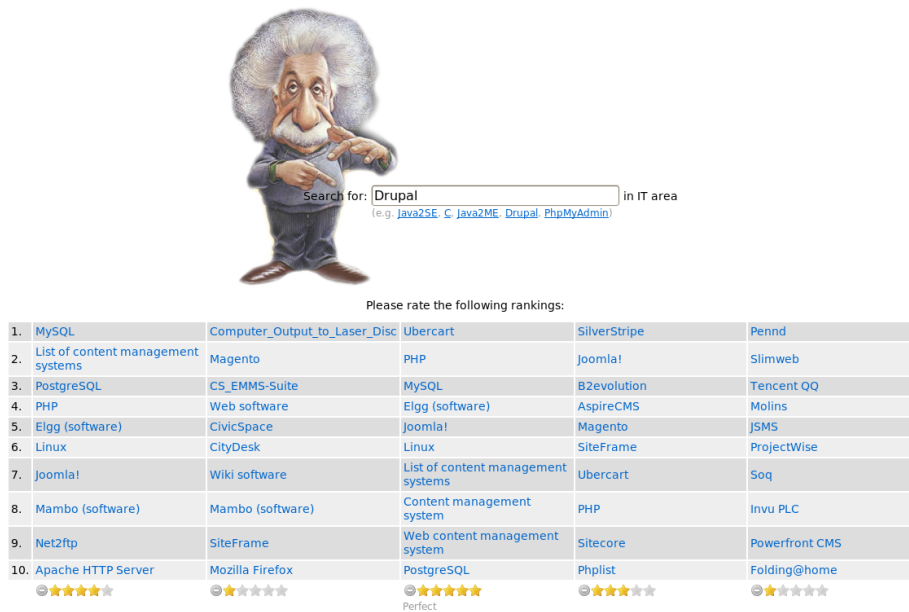


Figure 4: Screenshot of the evaluation system. The five columns show the results for, respectively, *Algo3*, *Algo4*, *DBpediaRanker*, *Algo2* and *Algo5*.

standard to query RDF datasets. Yet, if the considered dataset has a huge dimension, SPARQL will return as result of the query a long list of *not-ranked* resources. 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 list of results ranked w.r.t. the user’s query. In order to overcome this limit several PageRank-like [14] ranking algorithms have been proposed [6, 10, 12, 9]. 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 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 cannot be disregarded, instead it should be exploited in some way by the ranking process. *Swoogle* [6] is a semantic web search engine and a metadata search provider, which uses *OntologyRank*, 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* [10] 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 [10]. Our approach differs from [10] 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. [9] exploit the notion of naming authority, introduced by [12], 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 [9] loses its meaning in the case of a single huge source such as DBpedia. Mukherjea et al. in [13] presented a system to rank RDF resources inspired by [12]. As in the classical PageRank approach the relevance of a resource is decreased when there are a lot of outgoing links from that, nevertheless such an assumption seems not to be right in this case, as if an RDF resource has a lot of outgoing 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 cate-

<http://www4.wiwiw.fu-berlin.de/bizer/bookmashup/>,
<http://dbtune.org/>, <http://musicbrainz.org/>

gories. Based on this observation, URIs within the context can be interpreted as *hub* URIs. *TripleRank* [7], 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 [7] 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* [15], 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. [11] present a semantic search engine *NAGA*. It extracts information from several sources on the web and, then, finds relationships between the extracted entities. 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. In [17] the author proposes Semantic Link Network (*SLN*), a semantic data model to semantically link resources and derive implicit semantic links according to a set of relational reasoning rules. Inspired to this model, we adopt a different approach to elicit hidden knowledge and infer new meaningful links between resources in a RDF graph. In [18] the notion of *interactive semantics* is introduced as an “an open, self-organized and evolving social interactive system and its semantic image”. From this perspective our choice of rely on the *Linked Data* cloud and in particular on *DBpedia* well fits with the vision proposed by the author.

7. CONCLUSION AND FUTURE WORK

In this paper we presented a novel system for semantic tag generation and retrieval. We motivated our approach in the online advertising scenario, showing how exploiting semantic information in *DBpedia* it is possible both (i) to help advertisers in the process of keywords selection, and (ii) to enhance the ad selection process, displaying the most relevant ads w.r.t. the keywords entered in the search engine. 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. Moreover we are collecting ads about different domains and developing a platform to test our system in the real world.

8. REFERENCES

- [1] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
- [2] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. *DBpedia - a crystallization point for the web of data*. *Web Semantics: Science, Services and Agents on the World Wide Web*, July 2009.
- [3] A. Z. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, D. Metzler, L. Riedel, and J. Yuan. Online expansion of rare queries for sponsored search. In *Proceedings of the 18th International Conference on World Wide Web (WWW 2009)*, pages 511–520, 2009.
- [4] A. Z. Broder, M. Fontoura, V. Josifovski, and L. Riedel. A semantic approach to contextual advertising. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2007)*, pages 559–566, 2007.
- [5] R. Cilibrasi and P. Vitányi. The Google Similarity Distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.
- [6] 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.
- [7] T. Franz, A. Schultz, S. Sizov, and S. Staab. Triplerank: Ranking semantic web data by tensor decomposition. In *ISWC*, 2009.
- [8] E. Gabrilovich, A. Z. Broder, M. Fontoura, A. Joshi, V. Josifovski, L. Riedel, and T. Zhang. Classifying search queries using the web as a source of knowledge. *TWEB*, 3(2), 2009.
- [9] A. Harth, S. Kinsella, and S. Decker. Using naming authority to rank data and ontologies for web search. In *International Semantic Web Conference*, 2009.
- [10] A. Hogan, A. Harth, and S. Decker. ReConRank: A Scalable Ranking Method for Semantic Web Data with Context. 2006.
- [11] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *ICDE 2008*, 2008.
- [12] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. of 9th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 668–677, 1998.
- [13] 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.
- [14] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, 1998.
- [15] G. Tummarello, R. Delbru, and E. Oren. *Sindice.com: Weaving the Open Linked Data*. *The Semantic Web*, pages 552–565, 2008.
- [16] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.
- [17] H. Zhuge. Communities and emerging semantics in semantic link network: Discovery and learning. *IEEE Trans. on Knowl. and Data Eng.*, 21(6):785–799, 2009.
- [18] H. Zhuge. Interactive semantics. *Artificial Intelligence*, 174(2):190–204, 2010.