Discovering Unknown Connections – the DBpedia Relationship Finder

Jens Lehmann[1] lehmann@informatik.uni-leipzig.de
Jörg Schüppel[1] joergschueppel@web.de
Sören Auer[1,2] auer@seas.upenn.edu

[1]Universität Leipzig
Department of Computer Science
Johannisgasse 26
D-04103 Leipzig, Germany

[2]University of Pennsylvania
Department of Computer
and Information Science
Philadelphia, PA 19104, USA

Abstract: The Relationship Finder is a tool for exploring connections between objects in a Semantic Web knowledge base. It offers a new way to get insights about elements in an ontology, in particular for large amounts of instance data. For this reason, we applied the idea to the DBpedia data set, which contains an enormous amount of knowledge extracted from Wikipedia. We describe the workings of the Relationship Finder algorithm and present some interesting statistical discoveries about DBpedia and Wikipedia.

1 Introduction

Technologies based on Semantic Web standards are applied to various areas inside and outside the World Wide Web. A fundamental task is the creation and extension of ontologies, e.g. using the OWL[1] ontology language. In this work, we present a new user interface allowing to visualise connections in ontologies with large amounts of instance data.

The goal of the DBpedia Relationship Finder[2] is to provide a user interface to explore the huge DBpedia data set[ABK$^+$07] by providing a means to find connections between different objects. The background knowledge consists of all the facts, which have been extracted from Wikipedia, in particular the information extracted from infoboxes (see [AL07] for details). The resulting web application allows the user to enter two objects, which are described by articles in the English Wikipedia, and computes connections between them. The application makes heavy use of Web 2.0 concepts like AJAX. The connections are obtained by querying the RDF data of the underlying triple store. Therefore, the methods we propose and the user interface we develop can be used for arbitrary triple stores, as detailed in Section 5, but we will focus on DBpedia within this article. As a by-product of creating the Relationship Finder, we analysed the DBpedia RDF graph. We will present some interesting insights we gained.

[1]http://www.w3.org/2004/OWL
[2]available at http://wikipedia.aksw.org/relfinder/

Overall, the paper makes the following contributions:

- development of a new DBpedia user interface using Semantic Web and Web 2.0 techniques,
- statistical analysis of DBpedia and Wikipedia data,
- a new general RDF browsing interface.

The article is structured as follows: In Section 2 we give a brief overview of the DBpedia project. We then proceed to showing how we processed the obtained information in Section 3. The results of the preprocessing are used as input for the Relationship Finder and, furthermore, allow to derive some interesting statistics about DBpedia and therefore also Wikipedia. Section 4 describes the Relationship Finder algorithm and user interface. In Section 5 we give a different view of the DBpedia Relationship Finder as a general means to access the contents of RDF triple stores. We describe related work and conclude in Section 6.

2 The DBpedia Project

The DBpedia project[ABK$^+$07] is a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against Wikipedia and to link other datasets on the Web to Wikipedia data. The core of DBpedia is a method to extract RDF triples from the infobox templates used within Wikipedia articles (details can be found in [AL07]). Since Wikipedia authors developed templates which provide predefined information structures for a variety of domains the infobox dataset contains data for and relationships between entities from a multiplicity of knowledge domains. These include cities (4,872), music albums (35,190), people (19,834), books as well as information about special interest domains such as computer games (365), planes (527) or amphibians (736).

The sheer amount of multi-domain data of the infobox extraction dataset and the inability of existing tools to handle this amount of data inspired the development of the Relationship Finder and builds its basis. Besides information extracted from infoboxes, the DBpedia project also provides datasets containing various other information and metadata about Wikipedia articles, e.g. article abstracts, information about labels (in different languages), images and links related to articles and categories. All these datasets are provided for download as RDF dumps. They are accessible as linked data[BCH07] and available for querying in form of an SPARQL endpoint.

The DBpedia project also aims to be a hub for user interfaces visualizing DBpedia data for easy access and browsing by human users. The project comprises a query builder, a combined full-text and facet-based search interface and is browsable with linked-data browsers such as Tabulator[3] or Disco[4]. The availability of the DBpedia data in various

[3]http://www.w3.org/2005/ajar/tab

[4]http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/disco/

forms already stimulated many people to create mashups or specialized user interfaces. Despite its short time of existence, the DBpedia project already evolved into a crystallisation point for knowledge bases on the Web. The DBpedia datasets are interlinked with ontologies and knowledge bases such as Wordnet, Musicbrainz and Revyu.

3 Decomposition Algorithm and Statistical Discoveries

This section describes how we pre-processed the DBpedia RDF data to apply the Relationship Finder on it. Note, that the Relationship Finder can work even without this preprocessing step. However, some of its features will not be available in this case. The source code of the algorithms presented here and in the following sections are available within the DBpedia sourceforge project[5].

Algorithm 1: RDF Graph Decomposition.

```
Input: an RDF statements table (a set of triples)
Output: objects separated in components stored in a component table
create necessary database tables;
filter triples in the statements table and copy them in a table $T$;
initialise an empty queue $Q$;
clusterId = 0;
while $T$ is not empty do
    pick first object $O$ from $T$ and add it at the end of $Q$;
    write $O$ to component table;
    while $Q$ is not empty do
        find all objects obj, which are object or subject of a triple in $T$, which contains $O$
        as subject or object;
        forall $O' \in obj$ do
            if $O' \notin Q$ then
                add $O'$ at the end of $Q$;
                add $O'$ to component table;
            delete triples in $T$ containing $O'$;
        set $O$ to first object in $Q$;
    increment clusterId;
```

The goal of the pre-processing can be described as follows: We treat the extracted DBpedia infobox graph as an undirected graph and want to find its components, i.e. its maximal connected subgraphs. (Two objects are in the same component if and only if there exists a path between them.) Given two objects, this allows us to decide whether they are connected in the underlying RDF graph. If they are not connected, the Relationship Finder can terminate immediately. If they are connected, we want to be able to find a path between the

[5]http://dbpedia.svn.sourceforge.net/viewvc/dbpedia/relfinder/

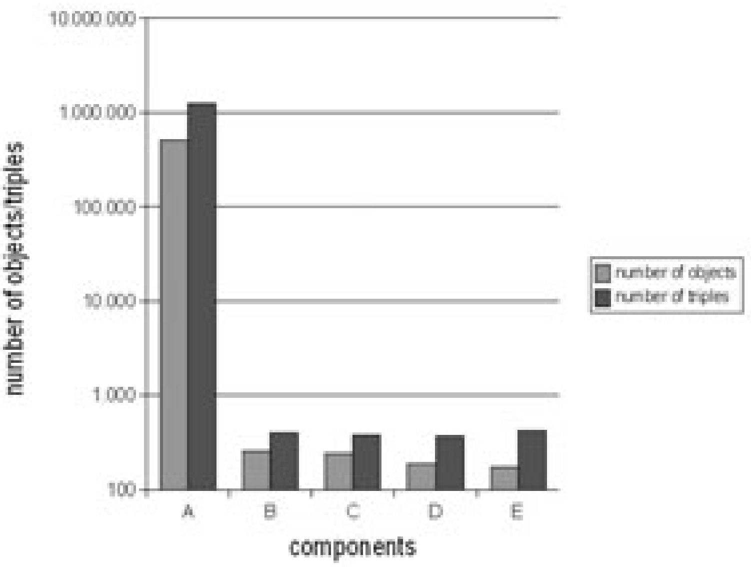


Figure 1: Number of triples and objects in the five largest components (ordered by number of objects).

two objects. While finding the shortest path is the computational most expensive part of the Relationship Finder algorithm, the pre-processing allows us to derive a (not necessarily shortest) path between two objects.

Algorithm 1 shows how we decompositioned the RDF graph. It works on an RDF statements table. Before applying the algorithm, we filter all triples, which we want to use, e.g. those not containing literal values, and copy them into a separate table. This filter can be configured to include or ignore certain types of triples if desired. The filtered DBpedia infobox data set still contains 1.5 million triples. We start from an arbitrary object and use a breadth first strategy to find all connected objects, i.e. all other objects within the component. The decomposition results are stored in database tables. For each object, we store its component id (the id of the component it belongs to), the minimum distance from our starting object within the component, and the object and property linking to the next object on the path to the starting object within the component. Please note, that the algorithm itself is not novel, but a straightforward application of existing techniques to RDF triple stores.

We determined that on average each DBpedia object has 5.67 outgoing connections, i.e. starting from an arbitrary object 5.67 other objects are directly connected. Considering outgoing connections of length two, 18 objects can be reached.

When running the cluster algorithm, we also generated some statistical information about the DBpedia components. Figure 1 shows the five largest components we obtained. Note the logarithmic scale on the y axis. We can see that almost all of the objects and triples are in the largest cluster. It accounts for 91% of all objects and 96% of all triples in the

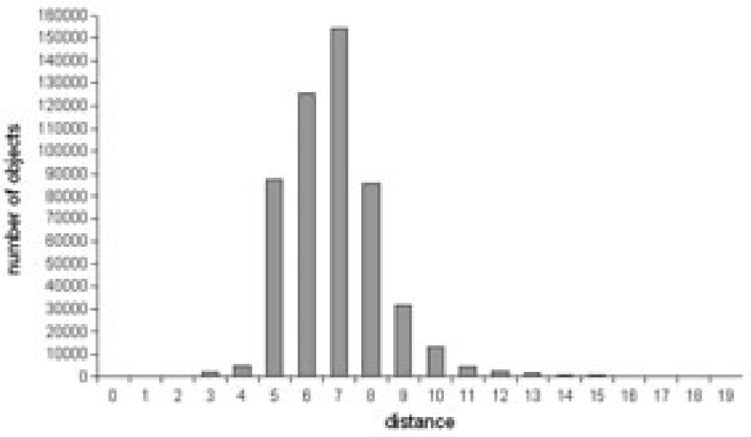


Figure 2: Number of objects with regard to the distance from a origin object.

DBpedia Infobox data set. This indicates that the DBpedia data set is densely connected, because starting from an arbitrary object in DBpedia (and therefore Wikipedia) this means you can reach almost any other object through properties in the DBpedia Infobox data set. This is one reason why it is interesting to construct a Relationship Finder, which allows to uncover these often interesting and surprising connections.

Figure 2 is another indicator of the density of the DBpedia RDF graph. As shown in Algorithm 1, we compute the components by starting with an arbitrary object. The figure shows the distance of any object in the main cluster from this starting object. Almost all of the objects have a distance between 5 and 9 from the starting object and are, thus, within a short distance from the starting object. The figure has to be interpreted cautiously, because it depends on the (randomly selected) starting object. A more comprehensive analysis is subject to further work.

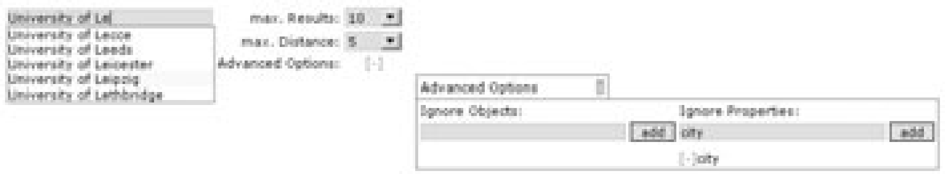
4 The Relationship Finder

This section describes the actual Relationship Finder web application based on the decomposition introduced before. We will first describe the user interface and then explain the underlying algorithm.

User interface. The Relationship Finder user interface is very intuitive. Initially, it contains a simple form to enter two objects, as well as a small number of options, and a list of previously saved queries. For entering objects, the user can utilize the autocompletion feature (see Figure 3), which is implemented using the freely available Scriptacolous JavaScript library[6]. While typing, the user is offered suggestions for the object he wants

[6]http://script.aculo.us/

to enter. The corresponding database queries are performed in the background and loaded into the displayed Web page using AJAX technology. After submitting the query by clicking the ”find relation” button, the Relationship Finder algorithm starts. First, the user is informed whether a connection between the objects exists. If such a connection exists, the user can, furthermore, preview a connection between the objects (see Figure 4). The details of this procedure are explained later.


University of Le
University of Lecce
University of Leeds
University of Leicester
University of Leipzig
University of Lethbridge

max. Results: 10
max. Distance: 5
Advanced Options: [-]

Advanced Options
Ignore Objects: add
Ignore Properties: city add
[-] city

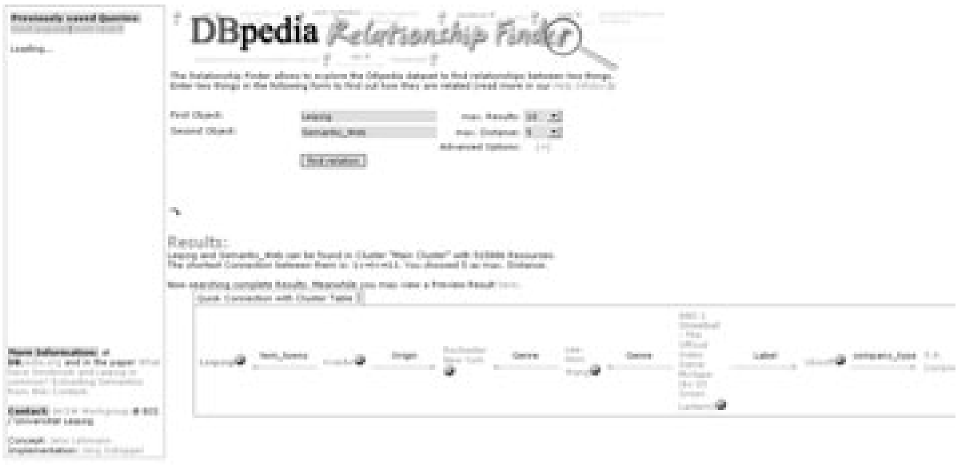
Figure 3: Autocompletion feature.

The preview connection does not have the guarantee to be the shortest available connection. For this reason, the Relationship Finder tries to find shorter solutions. When it has finished the computation, a configurable number of different connections is presented. Each connection is shown as a path where the leftmost part is the first entered object and the rightmost part is the second entered object. In between are the objects and properties, which connect these. Note, that the property arrows go in both directions, because we treat the underlying RDF graph as undirected.

Every object contains two additional buttons: The first one opens a box, which displays the available knowledge about this object. This information is retrieved from the statements database table using AJAX. The box parses information into user friendly formats, e.g. connections to other objects are shown as links, connection to images are directly displayed as image, and lists are recognised and displayed as such. All objects are transformed to links to the corresponding Wikipedia articles. The second button, which is associated with each object (and also each property in this case) is an ignore button depicted by a red cross. This allows to add objects and properties to an ignore list, i.e. the user states that in the next query he wants to ignore all connections containing these objects or properties. This list can also be edited by hand, again using autocompletion as a useful feature. Figure 5 shows a screenshot where both additional buttons are used.

After a query has been executed, the user can save it to make it available for other users. It is then displayed in the list of previously saved queries. This list can be ordered by popularity and query creation time. The results of these saved queries are cached, such that no significant server load is caused by executing these queries several time.

Technical Implementation. We assume that the components have been computed as described in Section 3. Another pre-processing step is to generate an undirected version of the statements table. This means that for each S-P-O triple, another O-P-S triple is written. This is done, because we consider the underlying RDF graph as undirected. The main reason why we are performing this as a pre-processing step (instead of the core algorithm) is efficiency.



DBpedia Relationship Finder

[illegible]

Results:

[illegible]

Figure 4: Precomputed connection between two objects.

Algorithm 2 shows the base structure of the Relationship Finder algorithm. Some parts will be explained in more detail in the next paragraphs.

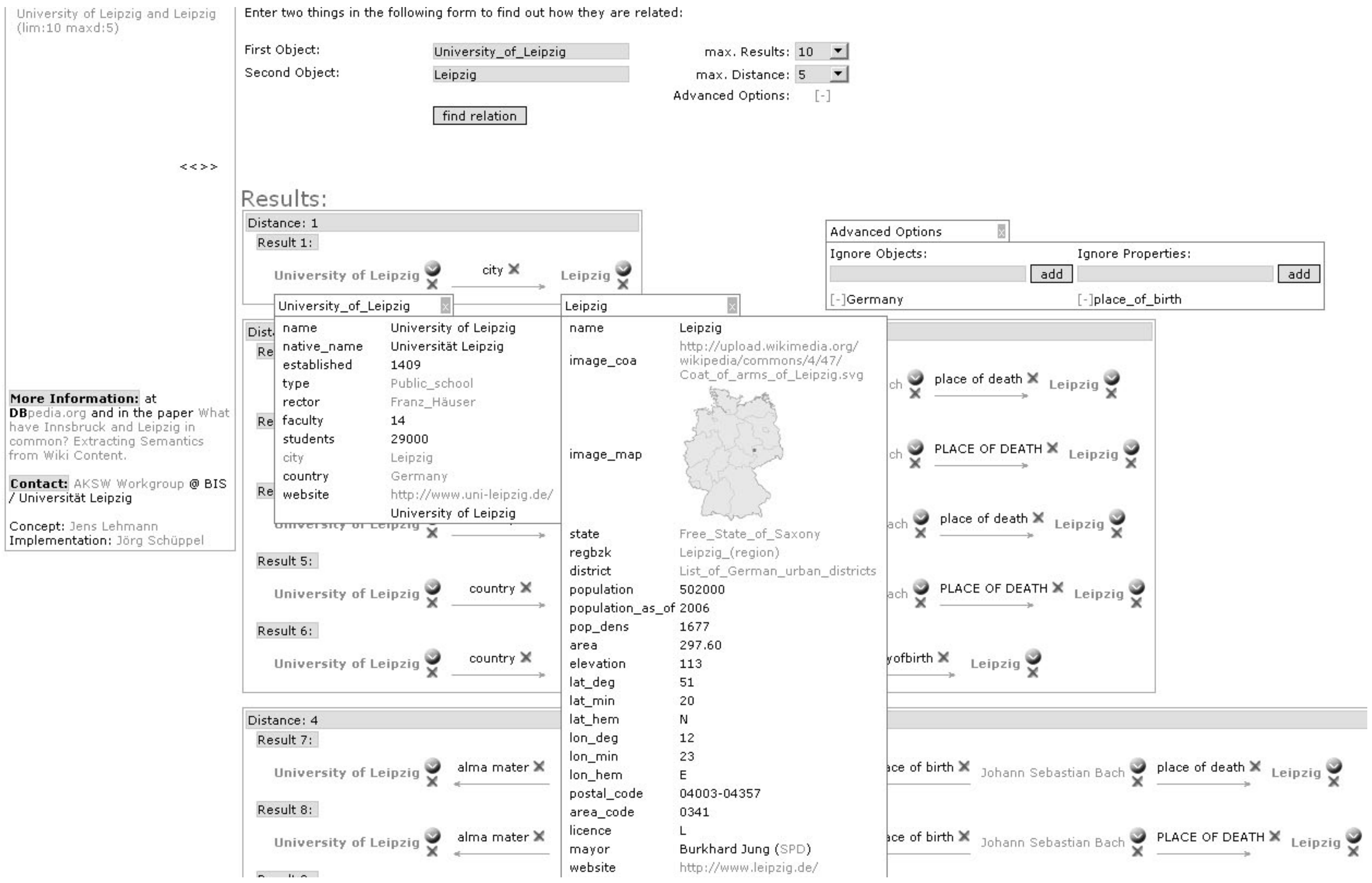
Line 2 states that a minimum and maximum distance between two objects O_1 and O_2 according to the components table are computed. This is done as follows: Let O_S be the starting object in the component of O_1 and O_2. From the components table we can obtain the two paths from O_1 to O_S and O_2 to O_S, respectively. The minimum distance *min* is then:

$$\mathit{min} = |\mathit{distance}(O_1, O_S) - \mathit{distance}(O_2, O_S)|$$

This follows from the fact that we used breadth first search in Algorithm 1, i.e. we know that the computed distances between an object and the starting object within a component are minimal. Say $\mathit{distance}(O_1, O_S) < \mathit{distance}(O_2, O_S)$ (without loss of generality), then the existence of a path with length smaller than *min* between O_1 and O_2 would imply that $\mathit{distance}(O_2, O_S)$ is not minimal, which is a contradiction.

Similarly, the maximum length is the sum of the distances. However, in this case we can give a better estimate. We can look for objects, which the paths from O_S to O_1 and O_S to O_2 have in common. If O_C is such an element, then $O_1 - \ldots - O_C - \ldots - O_2$ is a possible path from O_1 to O_2. We pick the common element, which minimises the length of such a path (due to O_S there is always a common element). The path we obtain is the one shown in the preview of the DBpedia Relationship Finder and its length is an upper bound of the length of the shortest path between O_1 and O_2.

The next interesting part of the algorithm is line 2, which generates the SQL database query to find the connections. The generated query contains JOIN operations corresponding to the current distance we are interested in. The underlying database systems usually optimise these operations, such that the JOINS are executed in an efficient order. However,



University of Leipzig and Leipzig
(lim:10 maxd:5)

<<>>

More Information: at DBpedia.org and in the paper What have Innsbruck and Leipzig in common? Extracting Semantics from Wiki Content.

Contact: AKSW Workgroup @ BIS / Universität Leipzig

Concept: Jens Lehmann
Implementation: Jörg Schüppel

Enter two things in the following form to find out how they are related:

First Object:	University_of_Leipzig	max. Results:	10
Second Object:	Leipzig	max. Distance:	5
		Advanced Options:	[-]

find relation

Results:

Distance: 1
Result 1:
University of Leipzig — city → Leipzig

Result 5:
University of Leipzig — country →

Result 6:
University of Leipzig — country →

Distance: 4
Result 7:
University of Leipzig ← alma mater — … ace of birth — Johann Sebastian Bach — place of death → Leipzig

Result 8:
University of Leipzig ← alma mater — … ace of birth — Johann Sebastian Bach — PLACE OF DEATH → Leipzig

… ch — place of death → Leipzig
… ch — PLACE OF DEATH → Leipzig
… ach — place of death → Leipzig
… ach — PLACE OF DEATH → Leipzig
… yofbirth → Leipzig

University_of_Leipzig

name	University of Leipzig
native_name	Universität Leipzig
established	1409
type	Public_school
rector	Franz_Häuser
faculty	14
students	29000
city	Leipzig
country	Germany
website	http://www.uni-leipzig.de/
	University of Leipzig

Leipzig

name	Leipzig
image_coa	http://upload.wikimedia.org/wikipedia/commons/4/47/Coat_of_arms_of_Leipzig.svg
image_map	
state	Free_State_of_Saxony
regbzk	Leipzig_(region)
district	List_of_German_urban_districts
population	502000
population_as_of	2006
pop_dens	1677
area	297.60
elevation	113
lat_deg	51
lat_min	20
lat_hem	N
lon_deg	12
lon_min	23
lon_hem	E
postal_code	04003-04357
area_code	0341
licence	L
mayor	Burkhard Jung (SPD)
website	http://www.leipzig.de/

Advanced Options

Ignore Objects:	Ignore Properties:
add	add
[-]Germany	[-]place_of_birth

Figure 5: Boxes with additional information and ignore list options in the Relationship Finder.

Algorithm 2: Workings of the DBpedia Relationship Finder.

```
Input: first object $O_1$, second object $O_2$, maximum distance $d_{max}$, maximum number of
       results $n$, ignore list of objects and predicates
1  if query has been saved then
2      load result from cache;
3  else
4      if $O_1$ and $O_2$ are in the same component then
5          compute minimum distance min and maximum distance max according to
           components table;
6          compute preview connection and display it;
7          set $d = min$;
8          set $m = 0$;
9          while $d < d_{max}$ and $m < n$ do
10             formulate SQL query for obtaining at most $(n - m)$ connections between $O_1$
               and $O_2$ of length $d$ without objects and properties in the ignore list;
11             if connections exist then
12                 display connections;
13                 $m = m +$ number of found connections;
14             increment $d$;
15             if $d = d_{max}$ then
                   Output: no connections within the specified maximum distance exist
16     else
           Output: no connection exists, objects in different components
```

depending on the query, these operations can still be very expensive for high distances, which is why we limit the distance between the two objects to 10. The SQL query is extended by constructs, which forbid double occurrences of objects and properties within a connection. Furthermore, the ignore lists are also taken into account here, i.e. we extend the query to disallow any of the objects and properties to be ignored in the connection.

5 The Relationship Finder as RDF Userinterface

This section gives some remarks about the use of the Relationship Finder for general RDF knowledge bases. As noted before, DBpedia is just one interesting application of the Relationship Finder. However, except for some DBpedia specific features (e.g. links to Wikipedia corresponding articles) it is not restricted to DBpedia. This is a brief overview of existing techniques for visualising ontology instance data:

- graphs: tools for navigating along RDF graphs
- tables: data organised in tabular form

- triples: data shown as basic triples
- timetables: usage of time oriented presentations, e.g. in personal organisers
- maps: usage of place oriented presentations, e.g. showing data in maps
- mashups: data collected from various sources and displayed together

The Relationship Finder extends this list by displaying a set of paths between two objects in an RDF graph of interest. A path here can be seen as a selection of interesting triples. The Relationship Finder is a general purpose user interface (such as graph or triple visualizations). It is especially suited for knowledge bases, which do not allow other visualization forms (such as graph or triple visualizations) due to their sheer amount of data.

6 Related Work, Conclusions, and Further Work

Related Work. We will first describe work related to the DBpedia project and afterwards work, which describes interfaces to RDF knowledge bases.

Apart from the DBpedia project, there have been other attempts to extract information from Wikipedia and make it available for further use. YAGO [SKW07] is an effort, which extracts 14 relations from the Wikipedia category system, Wikipedia redirects, and other sources of information within Wikipedia. Freebase[7] is a project by MetaWeb[8], which has the goal to build up a huge database of editable information. They used Wikipedia to reach an initial critical mass of information. Semantic MediaWiki [KVV05, VKV$^+$06] is an extension of the MediaWiki software, which is the Wiki software underlying Wikipedia. It allows to add structured data based on RDF to Wikis, which enables information reuse as well as enhanced search and browse facilities.

Techniques for discovering relationships between objects within knowledge bases were for example also developed in the course of the SemDis project[9]. In [AMHWA$^+$05] for instance, a flexible ranking approach is presented which can be used to distinguish more interesting and relevant relationships from less important ones. In [AMNR$^+$06], similar techniques were applied to address the problem of conflict of interest detection by analysing social networks.

Conclusions. We presented a novel RDF user interface, which is especially applicable to ontologies with large amounts of instance data. As one example for such an ontology, we used the DBpedia infobox data set. We implemented our approach and made it available online. We incorporated the feedback and feature requests we obtained in this application. The Relationship Finder uses a combination of existing algorithms in the background, AJAX technologies for providing a responsive and user-friendly interface,

[7]http://www.freebase.com
[8]www.metaweb.com/
[9]http://lsdis.cs.uga.edu/projects/semdis/

and numerous features like saving and caching queries, ignoring objects, and presenting additional information about objects.

Future Work. Possible lines of future work are to extend the Relationship Finder from DBpedia infoboxes to other parts of the DBpedia data set, to apply the Relationship Finder to other knowledge bases, and to improve our analysis of DBpedia/Wikipedia data.

References

[ABK+07] Sören Auer, Chris Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International Semantic Web Conference (ISWC)*, 2007. To appear.

[AL07] Sören Auer and Jens Lehmann. What Have Innsbruck and Leipzig in Common? Extracting Semantics fromWiki Content. In *Proceedings of the 4th European Semantic Web Conference (ESWC)*, pages 503–517, 2007.

[AMHWA+05] Boanerges Aleman-Meza, Christian Halaschek-Wiener, Ismailcem Budak Arpinar, Cartic Ramakrishnan, and Amit P. Sheth. Ranking Complex Relationships on the Semantic Web. volume 9, pages 37–44, 2005.

[AMNR+06] Boanerges Aleman-Meza, Meenakshi Nagarajan, Cartic Ramakrishnan, Li Ding, Pranam Kolari, Amit P. Sheth, Ismailcem Budak Arpinar, Anupam Joshi, and Tim Finin. Semantic analytics on social networks: experiences in addressing the problem of conflict of interest detection. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *WWW*, pages 407–416. ACM, 2006.

[BCH07] Christian Bizer, Richard Cyganiak, and Tom Heath. *How to publish Linked Data on the Web*. http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/, 2007.

[KVV05] Markus Krötzsch, Denny Vrandecic, and Max Völkel. Wikipedia and the Semantic Web - The Missing Links. In Jakob Voss and Andrew Lih, editors, *Proceedings of Wikimania 2005, Frankfurt, Germany*, 2005.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge - Unifying WordNet and Wikipedia. In *16th International World Wide Web Conference (WWW 2007)*, Banff, Canada, 2007.

[VKV+06] Max Völkel, Markus Krötzsch, Denny Vrandecic, Heiko Haller, and Rudi Studer. Semantic Wikipedia. In Les Carr, David De Roure, Arun Iyengar, Carole A. Goble, and Michael Dahlin, editors, *Proceedings of the 15th international conference on World Wide Web, WWW 2006*, pages 585–594. ACM, 2006.