RelFinder: Revealing Relationships in RDF Knowledge Bases

Philipp Heim[1], Sebastian Hellmann[2], Jens Lehmann[2], Steffen Lohmann[3], and Timo Stegemann[3]

[1] University of Stuttgart, Visualization and Interactive Systems, Universitätsstr. 38, 70569 Stuttgart, Germany
philipp.heim@vis.uni-stuttgart.de
[2] University of Leipzig, Agile Knowledge Engineering and Semantic Web, Johannisgasse 26, 04103 Leipzig, Germany
{hellmann,lehmann}@informatik.uni-leipzig.de
[3] University of Duisburg-Essen, Interactive Systems and Interaction Design, Lotharstr. 65, 47057 Duisburg, Germany
{steffen.lohmann,timo.stegemann}@uni-due.de

Abstract. The Semantic Web has recently seen a rise of large knowledge bases (such as DBpedia) that are freely accessible via SPARQL endpoints. The structured representation of the contained information opens up new possibilities in the way it can be accessed and queried. In this paper, we present an approach that extracts a graph covering relationships between two objects of interest. We show an interactive visualization of this graph that supports the systematic analysis of the found relationships by providing highlighting, previewing, and filtering features.

Keywords: semantic user interfaces, semantic web, relationship discovery, linked data, dbpedia, graph visualization.

1 Introduction

The Semantic Web enables answers to new kinds of user questions. Unlike searching for keywords in Web pages (as e.g. in *Google*), information can be accessed according to its semantics. The information is stored in structured form in knowledge bases using formal languages such as *RDF*[1] or *OWL*[2] and consisting of statements about real world objects like 'Washington' or 'Barack Obama'. Each object has a unique identifier (*URI*) and is usually assigned to ontological classes, such as 'city' or 'person', and an arbitrary number of properties that define links between the objects (e.g., 'lives in'). Given this semantically annotated and linked data, new ways to reveal relationships within the contained information are possible.

A common visualization for linked data are graphs, such as the *Paged Graph Visualization* [2]. In order to find relationships in these visualizations, users normally apply one of the following two strategies: They either choose a starting point

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/owl-features/

T.-S. Chua et al. (Eds.): SAMT 2009, LNCS 5887, pp. 182–187, 2009.

and incrementally explore the graph by following certain edges, or they start with a visualization of the entire graph and then filter out irrelevant data. Some more sophisticated solutions are based on the concept of faceted search. The tool *gFacet* [4], for instance, groups object data into facets that are represented by nodes and can be used to filter a user-defined result set. However, all these approaches require the user to manually explore the visualization in order to find relationships between two objects of interest. This kind of trial-and-error search can be very time consuming, especially in large knowledge bases that contain many data links.

As a solution to this problem, we propose an approach that automatically reveals relationships between two known objects and displays them as a graph. The relationships are found by an algorithm that is based on a concept proposed in [5] and that can be applied to large knowledge bases, such as *DBpedia* [1] or the whole *LOD-Cloud*[3]. Since the graph that visualizes the relationships can still become large, we added interactive features and filtering options to the user interface that enable a reduction of displayed nodes and facilitate understanding. We present an implementation of this approach – the *RelFinder* – and demonstrate its applicability by an example from the knowledge base *DBpedia* [1].

2 RelFinder

The RelFinder is implemented in Adobe Flex[4] and runs in all Web browsers with an installed Flash Player[5]. In the following, we first explain its general functionality before describing the involved mechanisms in more detail.

The search terms that are entered by the user in the two input fields in the upper left corner (Fig. 1, A) get mapped to unique objects of the knowledge base. These constitute the left and right starting nodes in the graph visualization (Fig. 1, B) that get then connected by relations and objects found in between them by the algorithm. If a certain node is selected all graph elements that connect this node with the starting nodes are highlighted forming one or more paths through the graph (Fig. 1, C). In addition, further information about the selected object is displayed in the sidebar (Fig. 1, D). Filters can be applied to increase or reduce the number of relationships that are shown in the graph and to focus on certain aspects of interest (Fig. 1, E).

2.1 Disambiguation

Ideally, the search terms that are entered by the user can be uniquely matched to objects of the knowledge base without any disambiguation. However, if multiple matches are possible (e.g., in case of homonyms, polysemes, or incomplete user input) the user is supported by a disambiguation feature. Generally, a list of objects with labels that enclose the search terms is already shown below the

[3] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

[4] http://www.adobe.com/products/flex

[5] The current version of the RelFinder is accessible at http://relfinder.dbpedia.org

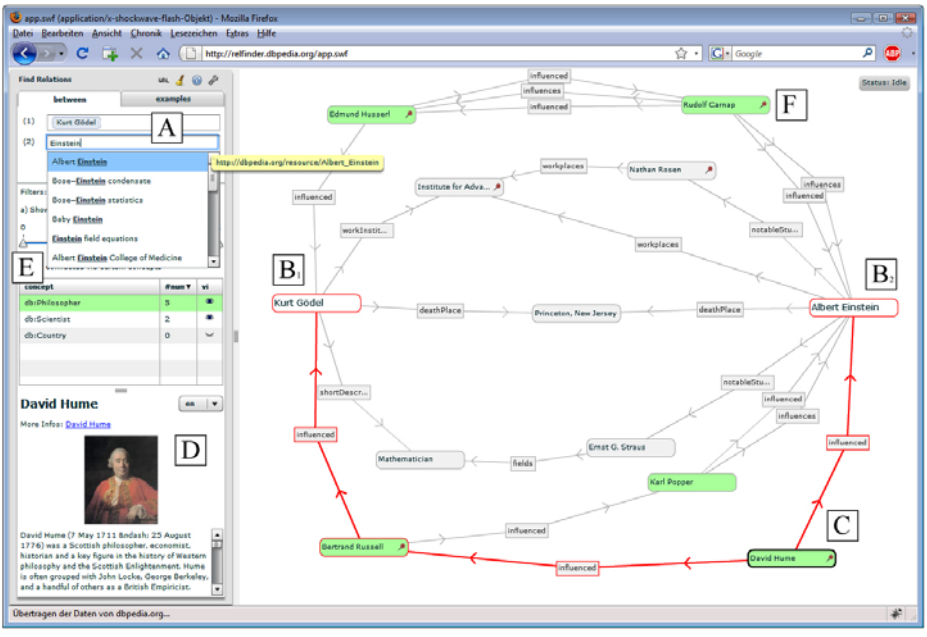

Fig. 1. Revealing relationships between Kurt Gödel und Albert Einstein

input box while the user enters the terms (Fig. 1, A). This disambiguation list results from a query against the SPARQL endpoint of the selected knowledge base. The following code shows the DBpedia optimized SPARQL query for the user input 'Einstein'[6]:

```
SELECT ?s ?l count(?s) as ?count WHERE {
?someobj ?p ?s .
?s <http://www.w3.org/2000/01/rdf-schema#label> ?l .
?l bif:contains '"Einstein"' .
FILTER (!regex(str(?s), '^http://dbpedia.org/resource/Category:')).
FILTER (!regex(str(?s), '^http://dbpedia.org/resource/List')).
FILTER (!regex(str(?s), '^http://sw.opencyc.org/')).
FILTER (lang(?l) = 'en').
FILTER (!isLiteral(?someobj)).
} ORDER BY DESC(?count) LIMIT 20
```

The disambiguation list is sorted by relevance using the 'count' value (or alternatively a string comparison if 'count' is not supported by the endpoint). 'count' is also used to decide if a user's search term can be automatically matched to an

[6] A configuration file allows to freely define the queried endpoint, the element that is queried (typically 'rdfs:label'), and properties that should be ignored. It is also possible to deactivate specific syntax elements, such as 'count' or 'bif:contains', in case a SPARQL endpoint does not support these.

object of the knowledge base or if a manual disambiguation is necessary. An automatic match is performed in one of the following two cases: 1) if the user input and the label of the most relevant object are completely equal, or 2) if the user input is contained in the label of the most relevant object and this object has a much higher count value than the second relevant object of the disambiguation list (ten times higher by default). Thus, the automatic disambiguation is rather defensive in order to prevent false matches.

If the user does not select an entry from the disambiguation list and if no automatic match is possible, the entries from the disambiguation list are shown again in a pop-up dialog that explicitly asks the user to provide the intended meaning of the search term by selecting the corresponding object.

2.2 Searching for Relationships

A query building process composed of several SPARQL queries searches for relationships between the given objects of interest. Since the shortest connection is not known in advance, the process searches iteratively for connections with increasing length, starting from zero. As a constraint, the direction of the property relations within each connection chain is only allowed to change once. We defined this constraint due to performance reasons and because multiple changes in the direction of the edges are difficult to be followed and understood by the user. If our objects of interests are a and b this results in the following search patterns:

$$\begin{gathered} a \rightarrow \cdots \rightarrow b \\ a \leftarrow \cdots \leftarrow b \\ a \rightarrow \cdots \rightarrow c \leftarrow \cdots \leftarrow b \\ a \leftarrow \cdots \leftarrow c \rightarrow \cdots \rightarrow b \end{gathered}$$

Thus, we are looking either for one-way relationships (first two lines) or those with an object c in between such that there is a one-way relationship each from a and from b to c or from c to a and b (last two lines). Note that c is not known in advance but found within the searching process.

The algorithm has several parameters: 1) It can be configured to suppress circles in extracted relationships. With the help of SPARQL filters, any object is only allowed to occur once in each connecting relationship. 2) Objects and properties can be ignored using regular expression patterns on their labels or URIs, which is useful if someone is not interested in certain objects or properties. More importantly, also structural relations between objects can be omitted, such as whether two objects belong to the same class or to the same part of a class hierarchy (i.e., ignoring *rdf:type* and *rdfs:subClassOf* properties). We decided to remove these by default, since they normally yield a multitude of relationships of minor interest which can be better explored in more traditional ways such a hierarchy browsers. 3) A maximum length of the returned relationships can be defined. 4) The SPARQL endpoint to use can be configured.

An exemplary SPARQL query that searches for relationships of the type *Kurt Gödel* $\leftarrow of1 \leftarrow c \rightarrow os1 \rightarrow$ *Albert Einstein* is given here (filter omitted):

```
SELECT * WHERE {
  db:Kurt_Gödel ?pf1 ?of1 .
  ?of1 ?pf2 ?c .
  db:Albert_Einstein ?ps1 ?os1 .
  ?os1 ?ps2 ?c .
    FILTER ...
} LIMIT 20
```

2.3 Graph Visualization

The found relationships are added one by one to the graph, beginning with the shortest (i.e., direct relationships and relationships with only one object in between, if there are any). All objects are visualized as nodes connected by edges that are labeled and directed according to the property relation they represent (Fig. 1, B). Since the labels of the edges are crucial for understanding the relationships they serve as flexible articulations in the force-directed layout [3], what reduces overlaps but cannot completely avoid them.

Interactive Features. To further reduce overlaps in the graph, we implemented a pinning feature that enables users to manually drag single nodes away from agglomerations and forces them to stay at the position they got dropped (pinned nodes are indicated by needle symbols as can be seen in Fig. 1, F). Especially in situations where many nodes are connected by many edges and thus are likely to overlap in the automatic layout, manual adjustments in combination with our pinning feature are helpful to produce an understandable graph layout that facilitates visual tracking. As already mentioned, visual tracking is additionally supported by the possibility of highlighting all paths that connect a selected node with the starting nodes (Fig. 1, C).

If a certain node is selected in the graph, further information about the corresponding object is displayed in the sidebar (e.g., in case of DBpedia these are a title, short abstract, and image extracted from Wikipedia, Fig. 1, D). Moreover, the ontological class an object belongs to is highlighted in the list that is shown in the sidebar (Fig. 1, E). Vice versa, all corresponding nodes in the graph are highlighted if an ontological class is selected from the list.

Filtering Options. The shown relationships can be filtered in two ways: 1) According to their length (i.e., the number of objects in between) and 2) according to the ontological classes the objects belong to. For instance, relationships consisting of several objects could be regarded as too far-fetched or objects belonging to certain classes might not be of interest for a user's goals (Fig. 1, E) and are therefore removed from the graph.

Filtering helps to reduce the number of displayed relationships in the graph and can hence prevent the graph from getting overly cluttered. For each search process, the filters are automatically set to initial values that avoid an overcluttered graph, if possible.

3 Conclusion and Outlook

We introduced an approach in this paper that uses properties in semantically annotated data to automatically find relationships between any pair of user-defined objects and visualizes them in a force-directed graph layout. The RelFinder can therefore save a lot of time which would otherwise be lost in searching these relationships manually. Since the amount of found relationships can be large, we additionally provide two types of filters that can be used to reduce the displayed relationships according to their length and the ontological classes they belong to and thus allow focusing on only a relevant part of the relationships. Together with a pinning feature that lets users rearrange the graph layout manually, we therefore provide a semi-automatic approach.

The basic mechanisms of the RelFinder work with every SPARQL endpoint and can therefore be applied to arbitrary knowledge bases with only little configuration effort. Since the SPARQL queries are composed on the client they are comparably independent of server routines and resources. We are currently testing the RelFinder on different knowledge bases in various scenarios.

Future work includes a comprehensive evaluation that further examines the potentials and benefits of querying structured knowledge bases compared to common Web search with respect to the discovery and analysis of relationships between certain objects of interests.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
2. Deligiannidis, L., Kochut, K., Sheth, A.: RDF data exploration and visualization. In: Proceedings of the ACM first workshop on CyberInfrastructure 2007, pp. 39–46. ACM Press, New York (2007)
3. Fruchterman, T., Reingold, E.: Graph drawing by force-directed placement. Softw. Pract. Exper. 21(11), 1129–1164 (1991)
4. Heim, P., Ziegler, J., Lohmann, S.: gFacet: A browser for the web of data. In: Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW 2008), pp. 49–58. CEUR-WS (2008)
5. Lehmann, J., Schüppel, J., Auer, S.: Discovering unknown connections – the DBpedia relationship finder. In: Proceedings of the 1st SABRE Conference on Social Semantic Web, CSSW (2007)