SPACE: SPARQL Index for Efficient Autocompletion

Kasjen Kramer, Renata Dividino, and Gerd Gröner

WeST, University of Koblenz-Landau, Germany

Abstract. Querying Linked Data means to pose queries on various data sources without information about the data and the schema of the data. This demo shows SPACE, a tool to support autocompletion for SPARQL queries. It takes as input SPARQL query logs and builds an index structure for efficient and fast computation of query suggestions. To demonstrate SPACE, we use available query logs from the USEWOD Data Challenge 2013.

1 Introduction

The linked data cloud is mainly accessible in two ways: SPARQL queries and direct RDF requests. Querying linked data on the Web using SPARQL is different to querying (relational) databases. First, linked data connect various data sources with heterogeneous data. Second, the schema and type statements are often unknown to the user or even missing in the data source. Thus, querying linked data means to pose queries on various data sources without information about the vocabulary and structure of the data.

In order to assist users when writing SPARQL queries, we show SPACE, a tool to support autocompletion of SPARQL queries. Autocompletion enables users to write queries fast. Existing approaches for query writing assistance make use of RDF datasets to extract query suggestions or they extract data source descriptions [1, 5] and/or relationships [2, 3]. Instead, we explore query logs available from SPARQL endpoints. We argue that previously executed queries are valuable information sources about the underlying data structure and schema of data sources. These queries reveal how resources are related and they reflect the user interests on resources and their relationships.

Our tool aims at enhancing usability of SPARQL query writing by providing suggestions of different possible query formulations to the user. Besides this, it enables fast and efficient computation of new suggestions. The computation of suggestions relies on an index structure for SPARQL queries. The SPACE index structure incorporates the structure and composition of graph patterns in SPARQL queries.

2 The SPACE Data Structure for Indexing SPARQL Queries

When a user writes a SPARQL query, SPACE aims to find the most similar queries available in the query logs in order to build new suggestions. A SPARQL query is a tuple defined as $Q = (A, V, G, P, M)$, where A is the set of prefix declarations (Line 1 in Fig. 1(a)), V is the output form (Line 2 in Fig. 1(a)). G refers to the RDF graph(s) being queried (Line 3 in Fig. 1(a)), P is a graph pattern (Line 5-8 in Fig. 1(a)) and M are query modifiers (Line 9 in Fig. 1(a)). In this work, we focus only on graph patterns. In that view, a query is composed only by its P. The core of SPACE is its index structure. The index structure is a graph, representing a set of SPARQL queries.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1>
SELECT ?person ?email
FROM <http://dig.csail.mit.edu/timbl/foaf.rdf>
WHERE{
{?personp foaf:name "John Doen"}
UNION
{?persion foaf:name "Peter Doen"}
?person foaf:mbox ?email
}LIMIT 50
```

(a) It returns the email of the persons named John Doen and Peter Doen

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person ?email
FROM <http://dig.csail.mit.edu/timbl/foaf.rdf>
WHERE {
{?person foaf:name "John Doen" }
UNION
{?person foaf:name "Sarah Carey"}
?person foaf:mbox ?email.
}LIMIT 50
```

(b) It returns the email of the persons named John Doen and Sarah Carey

Fig. 1. SPARQL Queries: Toy scenario

Definition 1 (SPACE Index). *The SPACE index $\mathcal{I}$ is a hierarchical index in form of a directed acyclic ordered graph $\mathcal{I} = (\mathcal{V}, \mathcal{E})$. Each vertex $v \in \mathcal{V}$ is associated with a level $l(v) \in 0, \ldots n-1$. Each edge $(v, v') \in \mathcal{E}$ leads to a vertex at a higher level $(l(v) < l(v'))$. The partial relation $\subseteq$ (set-inclusion) on the set $\mathcal{V}$ defines $v \subseteq v'$ for all $(v, v') \in \mathcal{E}$.*

According to Def. 1, the index structure has the following shape:

1. The vertices at the highest index level $(n-1)$ are represented by elements of the (pairwise disjoint) infinite sets I, B, L and V (IRIs, Blank nodes, literals and variables). Additionally, they represent the binary operators AND, UNION, OPT, FILTER, and GRAPH used to combine graph patterns. These vertices have only incoming edges.
2. The vertices from index level $n-2$ until index level 1 represent graph patterns, according to the recursive definition in [4]. A triple pattern is a graph pattern of the form $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$. If P1 and P2 are graph patterns then (P1 AND P2), (P1 OPT P2), and (P1 UNION P2) are also graph patterns. Given a SPARQL built-in condition R, then (P1 FILTER R) is a graph pattern. Finally, given a G $\in I$ or $\in V$, then (G GRAPH P) is a graph pattern.
3. The vertices from index level 1 represent SPARQL queries. Each query is composed by one graph pattern.
4. The (single) vertex, at the (lowest) level 0 (also called root vertex) represents a set of queries, e.g., all queries of a query log. This vertex has only outgoing edges.

Please note that, the number of graph patterns in the queries determine the height of the index tree. To illustrate our approach, Fig. 1(a) and Fig: 1(b) present two SPARQL queries. The first query searches for the email of the persons named John Doen and Peter Doen. The second one searches for the email of the persons named John Doen and Saray Carey. The SPACE index structure is shown in Fig. 2. The IRIs *foaf:name*, *foaf:mbox*, the literals *'John Coen'*, *'Peter Coen'* and *'Sarah Carey'*, the variables *?person* and *?email*, as well as the operator *AND* and *UNION* are represented by the nodes at the last level. The graph patterns are represented in the levels above. For instance, the triple pattern $t1$ is composed by the nodes *?person*, *foaf:name* and *'John Coen'*.

The process of searching for suggestions is done by sub-graph matching. Whenever there is a match of the query written by the user in the index graph, the tool is able to provide suggestions. The suggestions are ordered regarding to a popularity score. The popularity score represents the frequency of an element in the queries of the dataset.

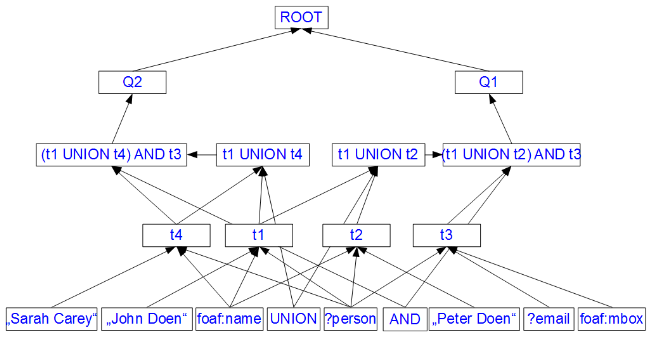


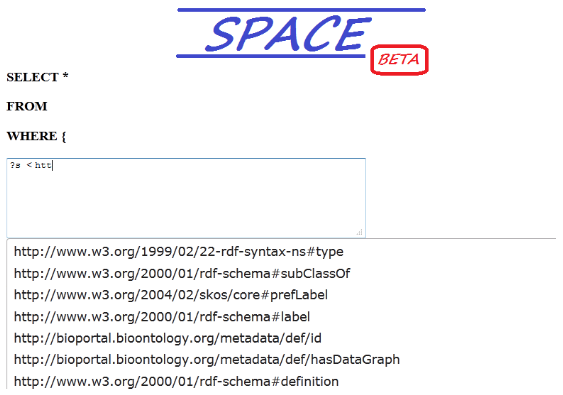
Fig. 2. SPACE Index of our example

When the user start writing a query, up the first symbols he writes, he gets suggestions. For instance, if the user starts with the symbols ⟨, then the tools suggest all possible URIs found in the graph' nodes. If the user writes "*?*", the tool searches for all the nodes representing a variable in the index. Given a variable, the possible follow-up suggestions are the predicates nodes, in our case, the predicates in "foaf:name" and "foaf:mbox", since they are the only predicates connected to variable nodes in the index graph. The tool only suggests (parts of) already observed queries. The more the user writes the smaller is the region where the query may be located in the index graph. Therefore, the longer the written query is, the more precise are the suggestions. The tool searches for the most similar queries in the query log in a bottom-up matching manner. For speeding up, an extra index for prefixes and namespaces is built. Please note that, nodes representing variables are not named and can be seen as just placeholders. The substitution method is used to check the equality of graph patterns.

3 SPACE in Use

Dataset: To demonstrate our tool, we collect queries from available query logs from the USEWOD Data Challenge 2013 [1] posted to SPARQL endpoints. In particular, the logs are from the two following sources: Open-BioMed.org.uk and BioPortal. The Open-BioMed.org.uk service offers gene expression search for Drosophila research, as well as drug discovery for the Alzheimer's disease. BioPortal provides access to commonly used biomedical ontologies.
Tool demonstration: The SPACE[2] is a web application tool written in Java and is based on the Jena framework. Its client-side is implemented in Javascript code. Fig. 3 shows a screenshot of the tool. SPACE is composed of two parts: (1) the not-editable part, representing an incomplete SELECT query and (2) the editable part, representing the graph pattern of the query. Suggestions are given to the user starting from the moment the user writes something in this field. The autocompletion functionality includes suggestion of IRIs such as classes and properties, of literals, of variables, of SPARQL binary operators as well as of namespaces and prefixes.

[1] USEWOD Data Challenge: `data.semanticweb.org/usewod/2013/challenge.html`
[2] SPACE: `http://west.uni-koblenz.de/Research/systems/SPACE`



SELECT *

FROM

WHERE {

```
?s <htt
```

http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/2000/01/rdf-schema#subClassOf
http://www.w3.org/2004/02/skos/core#prefLabel
http://www.w3.org/2000/01/rdf-schema#label
http://bioportal.bioontology.org/metadata/def/id
http://bioportal.bioontology.org/metadata/def/hasDataGraph
http://www.w3.org/2000/01/rdf-schema#definition

Fig. 3. SPACE screenshot

4 Conclusion and Outlook

In this paper, we have shown SPACE, a SPARQL editor that assists users when writing SPARQL queries. The tool is based on a hierarchical index structure of SPARQL queries, which enables fast computation of the most similar queries that are available in the query logs in order to generate new suggestions. So far, we have focused on the suggestions of query patterns.

We plan to proceed this research into three directions: (1) incorporate all operators of SPARQL and apply optimizations, (2) combine with approaches based on dataset statistics to improve recommendation, (3) conduct a user evaluation to get feedback from users to estimate which suggestion is intuitive with respect to human feeling.

References

1. S. Campinas, T. E. Perry, D. Ceccarelli, R. Delbru, and G.. Tummarello. Introducing RDF Graph Summary with application to Assisted SPARQL Formulation. In *DEXA*, 2012.
2. T. Gottron, A. Scherp, B. Krayer, and A. Peters. Lodatio: Using a schema-level index to support users in finding relevant sources of linked data. In *K-CAP'13*, 2013.
3. M. Jarrar and M. D. Dikaiakos. A Query Formulation Language for the Data Web. *IEEE Trans. on Knowledge and Data Engineering*, 24(5):783–798, 2012.
4. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, September 2009.
5. M. Zviedris and Barzdins G. ViziQuer: A Tool to Explore and Query SPARQL Endpoints. In *ESWC*, volume 6644 of *LNCS*, pages 441–445. Springer, 2011.