Live SPARQL Auto-Completion

Stéphane Campinas

Insight Centre for Data Analytics, National University of Ireland, Galway
stephane.campinas@insight-centre.org

Abstract. The amount of Linked Data has been growing increasingly. However, the efficient use of that knowledge is hindered by the lack of information about the data structure. This is reflected by the difficulty of writing SPARQL queries. In order to improve the user experience, we propose an auto-completion library[1] for SPARQL that suggests possible RDF terms. In this work, we investigate the feasibility of providing recommendations by only querying the SPARQL endpoint directly.

1 Introduction

The Linking Open Data movement has brought a tremendous amount of data available to the general user. The available knowledge spans a wide range of domains, from life sciences to films. However, using SPARQL to search through this knowledge is a tedious process, not only because of the syntax barrier but mainly due to the schema heterogeneity of the data. The expression of an information need in SPARQL is difficult due to the schema being generally unknown to the user as well as an heterogeneous of several vocabularies.

A common solution is for the user to manually gain knowledge about the data structure, i.e., what predicates and classes are used, by executing additional queries in parallel to the main one. The paper [3] proposes a "context-aware" *auto-completion* method for assisting a user in writing a SPARQL query by recommending schema terms in various position in the query. The method is context-aware in the sense that only essential triple patterns are considered for the recommendations. To do so, it leverage a data-generated schema. Instead, in this work we propose to bypass this need by executing live SPARQL queries in order to provide recommendations. Thus, this removes the overhead of pre-computing the data-generated schema. The proposed approach exposes a trade-off between the performance of the application and the quality of the recommendations. We make available a library[1] for providing data-based recommendations that can be used with other tools such as YASGUI [8].

In Section 2 we discuss related works regarding auto-completion for SPARQL. In Section 3 we present the proposed approach. In Section 4 we report an evaluation of the system based on query logs of DBpedia.

2 Related Work

Over the years, many contributions have been done towards facilitating the use of SPARQL, either visually [4], or by completely hiding SPARQL from the user [7]. In this work, we aim to help users with a knowledge of SPARQL by providing an auto-completion feature. Several systems have been proposed in this direction. Although

[1] Gosparqled: https://github.com/scampi/gosparqled

the focus in [1] is the visual interface, it can provide recommendations of terms such as predicates and classes. In [6] possible recommendations are taken from query logs. The system proposed in [5] provides recommendations based on the data itself, with a focus on SPARQL federation. Instead, we aim to make available an easy-to-use library which core feature is to provide data-based recommendations. In [3] an editor with auto-completion was developed that leverage a data-generated schema (i.e., a *graph summary*). We investigate in this work the practicability of bypassing the graph summary by relying only on the data.

3 Live Auto-Completion

We propose a data-based auto-completion which retrieves possible items with regards to the current state of the query. Recommended items can be predicates, classes, or even named graphs. Firstly, we indicate the position in the SPARQL query that is to be auto-completed, i.e., the *Point Of Focus* (POF), by inserting the character '<'. Secondly, we reduce the query down to its *recommendation scope* [3]. Finally, we transform the POF into the SPARQL variable *"?POF"* which is used for retrieving recommendations. The retrieved recommendations are then ranked, e.g., by the number of occurrences of an item.

Recommendation Scope. While building a SPARQL query, not all triple patterns are relevant for the recommendation. Therefore, we define the scope as the connected component that contains the POF. Figure 1a depicts a SPARQL query where the POF is associated with the variable "?s": it seeks possible predicates that occur with a ":Person" having the predicate ":name". Figure 1b depicts the previous SPARQL query reduced to its recommendation scope. Indeed, the pattern on line 4 is removed since it is not part of the connected component containing the POF.

```
1 SELECT * {
2   ?s a :Person;
3      :name ?name; < .
4   ?o a :Document
5 }
```

(a) A query with '<' as the POF

```
1 SELECT ?POF {
2   ?s a :Person;
3      :name ?name; ?POF [] .
4
5 }
```

(b) Scope of the query

Fig. 1: Query auto-completion

Recommendation Capabilities. The scope may include content-specific terms, e.g, resources and filters, unlike to [3] since the graph summary is an abstraction that captures only the structure of the data. Recommendations about predicates, classes and named graphs are possible as in [3]. In addition, the use of the data directly allows to provide recommendations for specific resources.

4 Evaluation

Systems. In this section, we evaluate the recommendations returned by the proposed system, that we refer to as "S1", against the ones provided by the approach in [3], which we refer to as "S2".

Settings. We compare the recommendations with regards to (1) the response-time, i.e., the time spent on retrieving the recommendations via a SPARQL query; and (2) the quality of the recommendations. A run of the evaluation consists of the following steps. First, we vary the amount of information retrieved via the "LIMIT" clause. Then, we compare the ranked TOP-10 recommendations against a gold standard. The ranking is based on the number of occurrences of a recommendation. The gold standard consists in retrieving recommendations directly from the data without the LIMIT clause, and retaining only the 10 most occurring terms. The TOP-10 of the gold standard and the system are compared using the *Jaccard* similarity. We consider that the higher the similarity, the higher the quality of recommendations.

Queries. We used the query logs of the DBpedia endpoint version 3.3 available from the USEWOD2013[2] dataset. The queries[3] were stripped of any pattern about specific resources, in order to keep only the structure of the query. In addition, we removed queries that contain more than one connected component. Queries are grouped according to their complexity, which depends on the number of triple patterns and on the number of star graphs. A group is identified by a string that has as many numbers as there are stars, with numbers separated by a dash '-' and representing the number of triple patterns in a star. For example, a query with two stars and one triple pattern each is then identified with *1-1*. This definition of query complexity exhibits the potential errors, i.e., a recommendation having zero-result, that a graph summary can have, as described in [2].

Graphs. We loaded into an endpoint the English part of the Dbpedia3.3[4] dataset, which consists of 167 199 852 triples. The graph summary consists of 29 706 051 triples, generated by grouping resources sharing the same set of classes.

Endpoint. We used a Virtuoso[5] SPARQL endpoint. The endpoint is deployed on a server with 32GB of RAM and with SSD drives.

Comparison. For each group of query complexity QC, we report in Table 1 the results of the evaluation, with $J1$ (resp., $J2$) the average Jaccard similarity for the system S1 (resp., S2); and $T1$ (resp., $T2$) the average response-time in ms for the system S1 (resp., S2). The reported values are the averages over 5 runs. We can see that as the LIMIT gets larger, the higher the Jaccard similarity becomes.Since the graph summary used in S2 is a concise representation of the graph structure, the data sample at a certain LIMIT value contains more terms than in S1. However, this impacts negatively on the quality of S2 as reflected by the values of J2. This shows the graph summary is subject to errors [2], i.e., zero-result recommendations. Nonetheless, it is interesting to remark that in S1 the recommendations can lead the query to an "isolated" part of the graph, from which the way out is through the use of "OPTIONAL" clauses. In S2, the graph summary allows to reduce this effect. The response-times for either system is similar, with S2 being slightly faster than S1. This indicates that directly querying the endpoint for recommendations is feasible. However, the significant difference in sizes between the graph summary and the original graph would become increasingly pre-dominant as the data grows.

[2] http://usewod.org/
[3] https://github.com/scampi/gosparqled/tree/master/eval/data
[4] http://wiki.dbpedia.org/Downloads33
[5] Virtuoso v7.1.0 at https://github.com/openlink/virtuoso-opensource

	$J1$	$J2$	$J1$	$J2$	$J1$	$J2$	$J1$	$J2$	$J1$	$J2$	$J1$	$J2$	$J1$	$J2$	$J1$	$J2$
QC	2		3		4		5		6		9		10		1-1		1-2	
10	0.12	0.12	0.17	0.21	0.15	0.21	0.16	0.19	0.14	0.16	0.17	0.19	0.16	0.19	0.11	0.09	0.19	0.18
100	0.15	0.17	0.28	0.26	0.27	0.28	0.28	0.29	0.24	0.26	0.25	0.26	0.25	0.27	0.12	0.11	0.24	0.22
500	0.24	0.27	0.34	0.29	0.34	0.30	0.36	0.35	0.38	0.31	0.42	0.27	0.43	0.26	0.15	0.18	0.29	0.29
QC	1-3		1-4		1-5		2-2		3-4		1-1-2		1-1-3		1-1-4			
10	0.62	0.64	0.23	0.22	0.15	0.19	0.17	0.17	0.15	0.06	0.55	0.38	0.50	0.49	0.38	0.43		
100	0.62	0.60	0.38	0.32	0.24	0.32	0.19	0.19	0.24	0.10	0.57	0.39	0.53	0.52	0.44	0.40		
500	0.62	0.59	0.60	0.34	0.25	0.29	0.25	0.22	0.21	0.12	0.57	0.40	0.55	0.51	0.47	0.46		

	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$	$T1$	$T2$
QC	2		3		4		5		6		9		10		1-1		1-2	
10	107	81	119	82	127	81	129	82	144	85	314	197	688	468	97	79	103	80
100	108	81	180	84	147	95	202	86	173	88	311	198	701	458	122	84	140	83
500	141	91	192	96	144	79	172	99	149	101	337	207	701	467	127	89	133	111
QC	1-3		1-4		1-5		2-2		3-4		1-1-2		1-1-3		1-1-4			
10	101	108	108	87	104	93	102	80	114	83	107	391	106	87	105	87		
100	103	105	102	94	105	84	106	80	142	85	115	385	112	89	105	96		
500	126	105	141	92	136	97	158	94	137	117	126	400	133	99	139	102		

Table 1: Average Jaccard similarity ($J1$ for system S1 and $J2$ for S2) and response-times in ms ($T1$ for system S1 and $T2$ for S2) for each group of query complexity QC, and with the LIMIT varying from 10 to 500. The reported values are the averages over 5 runs.

Acknowledgement

This material is based upon works supported by the European FP7 projects LOD2 (257943).

Bibliography

[1] Ambrus, O., Mller, K., Handschuh, S.: Konduit vqb: a visual query builder for sparql on the social semantic desktop

[2] Campinas, S., Delbru, R., Tummarello, G.: Efficiency and precision trade-offs in graph summary algorithms. In: Proceedings of the 17th International Database Engineering & Applications Symposium. pp. 38–47. IDEAS '13, ACM, New York, NY, USA (2013)

[3] Campinas, S., Perry, T.E., Ceccarelli, D., Delbru, R., Tummarello, G.: Introducing rdf graph summary with application to assisted sparql formulation. In: Proceedings of the 2012 23rd International Workshop on Database and Expert Systems Applications. pp. 261–266. DEXA '12, IEEE Computer Society, Washington, DC, USA (2012)

[4] Clark, L.: Sparql views: A visual sparql query builder for drupal. In: International Semantic Web Conference. pp. –1–1 (2010)

[5] Gombos, G., Kiss, A.: Sparql query writing with recommendations based on datasets. In: Yamamoto, S. (ed.) Human Interface and the Management of Information. Information and Knowledge Design and Evaluation, Lecture Notes in Computer Science, vol. 8521, pp. 310–319. Springer International Publishing (2014)

[6] Kramer, K., Dividino, R.Q., Gröner, G.: Space: Sparql index for efficient autocompletion. In: Blomqvist, E., Groza, T. (eds.) International Semantic Web Conference. CEUR Workshop Proceedings, vol. 1035, pp. 157–160. CEUR-WS.org (2013)

[7] Lehmann, J., Bühmann, L.: Autosparql: Let users query your knowledge base. In: Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part I. pp. 63–79. ESWC'11, Springer-Verlag, Berlin, Heidelberg (2011)

[8] Rietveld, L., Hoekstra, R.: Yasgui: Not just another sparql client. In: SALAD@ESWC. pp. 1–9 (2013)