On Enhancing Visual Query Building Over KGs Using Query Logs (Short Paper)

Vidar Klungre[1], Ahmet Soylu[2,3], Martin Giese[1], Arild Waaler[1], and Evgeny Kharlamov[1,4]

[1] University of Oslo, Oslo, Norway
{vidarkl, martingi, arild, evgeny.kharlamov}@ifi.uio.no
[2] Norwegian University of Science and Technology, Gøvik, Norway
ahmet.soylu@ntnu.no
[3] SINTEF Digital, Oslo, Norway
[4] University of Oxford, Oxford, United Kingdom
evgeny.kharlamov@cs.ox.ac.uk

Abstract. Knowledge Graphs have recently gained a lot of attention and have been successfully applied in both academia and industry. Since KGs may be very large: they may contain millions of entities and triples relating them to each other, to classes, and assigning them data values, it is important to provide endusers with effective tools to explore information incapsulated in KGs. In this work we present a visual query system that allows users to explore KGs by intuitively constructing tree-shaped conjunctive queries. It is known that systems of this kind suffer from the problem of information overflow: when constructing a query the users have to iteratively choose from a potentially very long list of options, sich as, entities, classes, and data values, where each such choice corresponds to an extension of the query new filters. In order to address this problem we propose an approach to substantially reduce such lists with the help of ranking and by eliminating the so-called deadends, options that yield queries with no answers over a given KG.

1 Motivation and Overview

Motivation. Knowledge Graphs (KGs) are collections of interconnected entities annotated with classes and data values, which have become powerful assets for enhancing search and are now widely used in both academia and industry. Prominent examples of large-scale knowledge graphs include Yago [23], Google's Knowledge Graph [1], that are used by search engines, and Siemens [16] and Statoil [15] corporate KGs.

Many existing knowledge graphs are either available as Linked Open Data, or they can be exported as RDF datasets [4] enhanced with OWL 2 ontologies [3] capturing the relevant domain background knowledge. SPARQL [11] has become the standard language for querying KGs stored as RDF datasets with OWL 2 ontologies, and an increasing number of applications offer SPARQL endpoints to

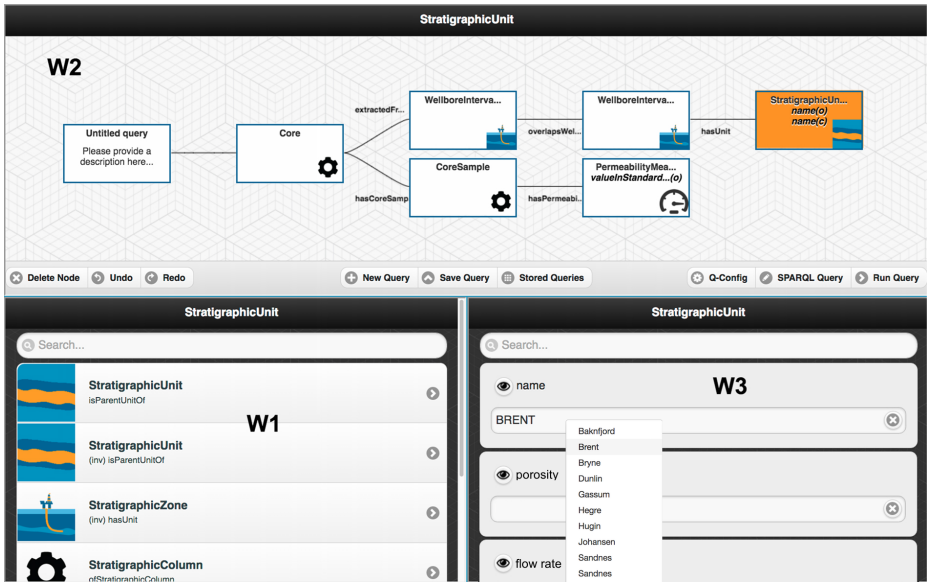

Fig. 1: Example visual query system OptiqueVQS

access KGs. Writing SPARQL queries, however, requires some proficiency in the query language and is not well-suited for the majority of users [25,12]. Thus, an important challenge that has attracted a great deal of attention in the Semantic Web community is the development of simple yet powerful query interfaces for non-expert users [19,2,9,10,26,7].

Visual Query Building. An important class of such interfaces for KGs is *visual query building* systems [20], where the users can construct queries by combining classes and properties offered by the system, and by setting constraints on classes and/or properties by selecting appropriate values offered by the system. Consider in Figure 1 a prominent example of a system for visual query building, OptiqueVQS [22,21], that we developed in tight collaboration with companies such as Statoil and Siemens. In the following example we illustrate the query formulation process in OptiqueVQS that should help the reader in understanding the mode of interaction between the users and visual query systems and will also help us to clarify challenges that we address in this work.

Example 1. In Figure 1 one can see a real world query that was constructed by a Statoil engineer when we conducted a user study. The engineer constructed the following query:

> *Give me all the wellbore cores extracted from a wellbore interval, such that it overlaps with another wellbore interval whose stratigraphic unit is named 'BRENT', along with all the permeability measurements of their samples and the values of these measurements in standard unit.*

Now we explain OptiqueVQS in more details. The system has three main widgets: The first widget (W1 in bottom-left of Figure 1) is menu-based and it allows the user to navigate through concepts of an ontology by selecting relationships between them. The second widget (W2 on top of Figure 1) is diagram-based and it presents typed variables as nodes and object properties as arcs and gives an overview of the query formulated so far. The third widget (W3 in bottom-right of Figure 1) is form-based and it presents the attributes of a selected concept for selection and projection operations.

W1 initially lists all the concepts in the ontology and a user starts formulating a query by selecting a starting concept. The concept chosen from W1 becomes the active node (i.e., pivot) and appears in W2 as a variable node. W1 then lists concept - object property pairs pertaining to the pivot, since there is now an active node. The user can continue adding more typed variables into the query by selecting a pair from W1. The selected concept-object property pair is added to the query over the pivot, and the formulated query is presented as a tree in W2. The concept from the last chosen pair automatically becomes the active node (i.e., pivot), and the active node can be changed by clicking on the corresponding variable node in W2. The user can constrain attributes (i.e., using the form elements) and/or select them for output (i.e., using the "eye" icon) through W3. The user can also refine the type of a variable node through a special multi-select form element, called "Type", in W3. (Not included in screenshot.)

To run the constructed query, the user clicks the "Run Query" button and the system allows to see sample query results and to manipulate them, e.g., to apply sorting and aggregation operations, see the widget W4 in Figure 2. Moreover, the user can view and interact with the query in textual SPARQL mode (i.e., textual and visual modes are synchronised). In the top part of Figure 2 one can see the SPARQL version of our example query from Figure 1. The user can also save, modify, and load queries. □

Information Overload Problem. An important challange for visual query systems applied in the context of KGs is *information overload* [24,14]: a number of options for query construction that a visual query system offers to a user is comparable to the size of data over which the query is constructed. Since large scale KGs contain millions of entities, data values, and up to hundreds of classes, the number of options hampers the query construction process. Our OptiqueVQS system is not an exception and it also suffers from the information overload problem. Indeed, consider in Figure 1, even for a small ontology the number of suggestions given in W1 for a pivot concept would require a user to scroll down in the list several times due to high number of potential concept-object property pairs and reasoning effect. The latter requires propagating concept restrictions upwards and downwards in the hierarchy and results in new concept-object property pairs for a given concept originating from its parent and child concepts [8]. The same applies to W3 not only in terms of data properties being suggested but also potential values being offered for each property. For example,

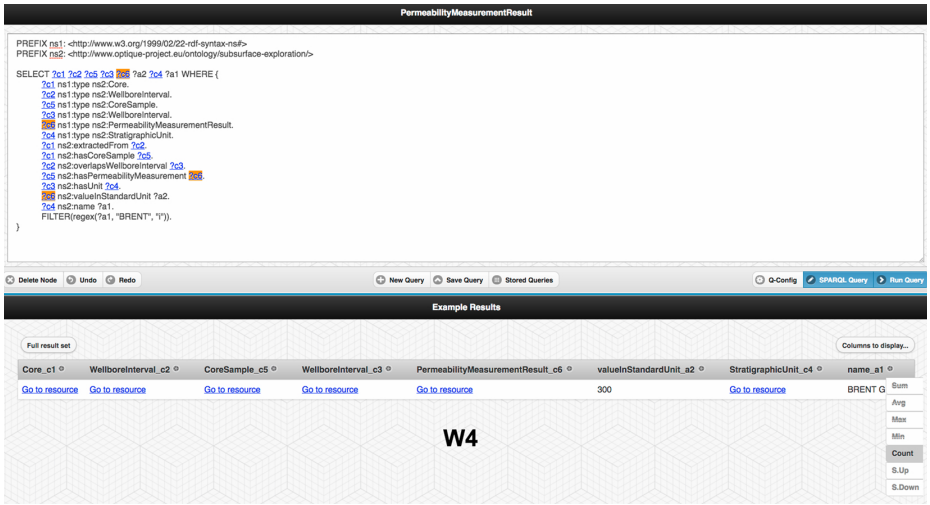

Fig. 2: Query from Example 1 in SPAQRL, presented in OptiqueVQS

the potential values offered for the name attribute in the example is too extensive: Statoil database has several thousands of names and their variations.

Query Logs in Enhancing Query Building. In this work we propose to exploit query logs, that is, queries that the users have constructed when they used the visual query system, to enhance the query building process and reduce the information overload. Our first idea is to exploit query logs in ranking and top-k computation, that is, we show to the user only those components for query construction that occur often enough in the logs. Our second idea is to show to the user only those components for query construction that are not deadends, that is, if the user relies on any of them in query construction then the resulting query will not return empty answers over the underlying dataset. As we discuss later in the paper, deadend elimination is computationally demanding and we rely on query logs to reduce the cost of such elimination.

This is a preliminary study, we are still developing our approaches. In the following section we give more details on our ideas.

2 Our Approach and Further Directions

The main hypothesis behind our approach is that the queries over KGs that are often used by users during query construction are more important than the ones used less often or not used at all. We analyse the frequency of usage for queries and their fragments from query logs and then use the frequency to optimise suggestions of elements of KGs that we suggest to users during query construction sessions.

Queries and Query Logs. We assume that the reader is familiar with the basic notions of RDF and SPARQL, queries, databases, query evaluation and refer the reader to [3,11] for further details. In this work we consider only tree-shaped conjunctive queries over classes, object properties, data properties, entities, data values, and variables since they are supported by OptiqueVQS [21]. Given a database D and a query Q, with $ans(Q, D)$ we denote the answer set of Q over D. A query log $\mathcal{Q}$ is a set of queries. A query pattern P is a query that contains no entities or data values. Given a query Q, a query pattern for Q, denoted $p(Q)$, is Q where all entities and data values are substituted with fresh variables in such a way that the same entities are and the same values are substituted with the same variables. With $Q_1 \subseteq Q_2$ and $P_1 \subseteq P_2$ we denote that the query Q_1 is a subquery of Q_2 and that the pattern P_1 is a subquery of P_2. Finally, a *query suggestion* E is a query that consists of one atom. Then, given a query Q, with $Q \wedge E$ we denote a query obtained by extending Q with E by adding E to Q with the conjunction.

Ranking Based on Query Logs. Given a query log $\mathcal{Q}$ and a query pattern P, we define the conditional probability of P wrt to $\mathcal{Q}$ as the frequency of queries from $\mathcal{Q}$ whose patterns contain P as in Equation (1) on the left, where $|\cdot|$ as usual denotes the cardinality of a set. Now we define how a given pattern T is 'important' for another pattern P wrt the query log $\mathcal{Q}$ as the ranking function in Equation (1) on the right.

$$Pr(P \mid \mathcal{Q}) = \frac{|\{Q \in \mathcal{Q} \mid P \subseteq p(Q)\}|}{|\mathcal{Q}|}, \qquad r(T \mid P, \mathcal{Q}) = \frac{Pr(T \cap P \mid \mathcal{Q})}{Pr(P \mid \mathcal{Q})} \quad (1)$$

Let Q be a query that is constructed by a user, $\mathcal{Q}$ be a query log, and E a query suggestion. Finally, we define the ranking of a suggestion E for Q wrt $\mathcal{Q}$ as the average rank of $Q \wedge E$ for patterns in $\mathcal{Q}$, that is, $r_{Q,\mathcal{Q}}(E) = \sum_{Q' \in \mathcal{Q}} r(p(Q \wedge E) \mid p(Q'), \mathcal{Q})/|\mathcal{Q}|$. In other words, the rank of a suggestion E is defined via the importance of the pattern of E to the pattern of Q wrt the query log $\mathcal{Q}$.

This approach can be further extended by incorporating the semantic distance between concepts and properties involved as a cofactor into the ranking function, so that that queries from $\mathcal{Q}$ that are semantically distant from Q contribute less to the ranking. For example, Huang et al. [13] suggest a similarity measure using the depth of compared concepts and properties and their least common ancestors from the root of hierarchy to compute similarity between concepts and properties and combining them to compute similarity between triple patterns, hence queries.

Deadend Elimination Based on Query Logs. In this approach we eliminate so called *contextual deadends* [24,5], that is, query extensions E for a query Q such that the query $Q \wedge E$ evaluated over a given database D gives the empty set of answers. For us both Q and D form the *context* for E. Given a set of suggestions $\mathcal{E}$ and a context Q and D, a naive way to eliminiate deadends

would be to go over $\mathcal{E}$, to pick $E \in \mathcal{E}$ one after another, to check whether $ans(Q \wedge E, D) = \emptyset$, and to delete from $\mathcal{E}$ all E for which it is the case. The suggestions that remain in $\mathcal{E}$ at the end of this procedure would not be deadends. A disadvantage of this approach is that such procedure requires to run $|\mathcal{E}|$ queries over D and this is computationally demanding for a large $|\mathcal{E}|$. Indeed, even when $|\mathcal{E}| = 10,000$ modern RDF backends would not be able to process that many queries in time acceptable for an interactive system, that is, in time that many users would tollerate to wait [5]. Indeed, it will take on average 15 seconds for such well known systems as Sesame [6][5] Stardog [18][6], and RDFox [17][7] to run that many queries. At the same time, $|\mathcal{E}| = 10,000$ can easily occur in practice, indeed, observe that DBpedia contains more that 6 million entities and many of them can be potentially relevant for being suggested, e.g., via the *linked-to* object property[8]. Even for $|\mathcal{E}| = 1,000$ the average query evaluation time is about 3 seconds [5] and DBpedia contains more than $2,700$ data and object properties that can potentially be relevant and suggested to a user.

In order to speedup deadend elimination we propose to rely on query logs. Our idea is, given a database D and a query log $\mathcal{Q}$, to select among the queries in the log the most promising query patterns P, that is, the most frequent ones, and to pre-materialise answers $ans(P, D)$ for each Ps as databases D_P in a 'smart way' (see details below). Then, given a query Q and a set of suggestions $\mathcal{E}$ one can verify for each $E \in \mathcal{E}$ the emptiness of the answer set $ans((P \cap Q) \wedge E, D_P)$ instead of $ans(Q \wedge E, D)$ and *do not* suggest those E to the user that pass the emptiness test. One can show that this procedure is an approximation of the naive one described above, that is, there are cases when (i) $ans((P \cap Q) \wedge E, D_P) = \emptyset$, but $ans(Q \wedge E, D) \neq \emptyset$, and (ii) $ans((P \cap Q) \wedge E, D_P) \neq \emptyset$, but $ans(Q \wedge E, D) = \emptyset$. One can partially address Cases (i) by the way that D_P is materialised: one can materialise $ans(P', D)$, for P' that slightly extends P with all possible extensions at the output variable. One can show that this approach does not affect Cases (ii) and our preliminary experiments show that this also helps to significantly reduce the number of Cases (i) in practice.

Further Directions. Our current research effort is targeted towards evaluating the approaches for ranking and deadend elimination as well as towards combining and improving them. For example, the approach that we see as promising is to first compute top-k suggestions given a query partially constructed by a user and a query log, and then to eliminate the deadends among the top-k. Our preliminary evaluation of our approximate approach to deadend elimination shows

[5] Sesame is a widely-used Java framework for processing RDF data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions.

[6] Stardog is a Java-based triple store providing reasoning support for all OWL 2 profiles as well as a SPARQL implementation.

[7] RDFox is an in-memory RDF triple store that supports shared memory parallel Datalog reasoning. It is written in C++ and comes with a Java wrapper allowing for a seamless integration with Java-based applications.

[8] https://wiki.dbpedia.org/dbpedia-version-2016-04

that we can eliminate up to 80% of deadends using pre-materialised views that contain about 10% of the original data. This gives us a significant improvement in terms of time over the naive approach without a dramatic compromise on the quality of deadend elimination.

3 Conclusion

The information overload is a challenging problem that is vital for the query building over large scale KGs. In this short paper we presented our preliminary work on enhancing visual query building over KGs by addressing the overload problem with the help of query logs. We presented a ranking model and an way to eliminate deadends, as well as a discussion how they can be combined and further improved. We are excited to present our work to the Semantic Web community.

Acknowledgements This work is partially funded by EU H2020 TheyBuyForYou (780247) project.

References

1. Google's Knowledge Graph. http://www.google.co.uk/insidesearch/features/search/knowledge.html.
2. iSPARQL QBE. http://dbpedia.org/isparql/.
3. W3C: OWL 2 Web Ontology Language. http://www.w3.org/TR/owl2-overview/.
4. W3C: Resource Description Framework (RDF). http://www.w3.org/RDF/.
5. Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. Faceted search over rdf-based knowledge graphs. *J. Web Sem.*, 37-38:55–74, 2016.
6. Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proc. of ISWC*, pages 54–68, 2002.
7. Enrico Franconi, Paolo Guagliardo, Marco Trevisan, and Sergio Tessaris. Quelo: an Ontology-Driven Query Interface. In *DL*, 2011.
8. Bernardo Cuenca Grau, Martin Giese, Ian Horrocks, Thomas Hubauer, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Michael Schmidt, Ahmet Soylu, and Dmitriy Zheleznyakov. Towards query formulation, query-driven ontology extensions in OBDA systems. In *OWLED*, 2013.
9. Florian Haag, Steffen Lohmann, Stephan Siek, and Thomas Ertl. Visual querying of linked data with QueryVOWL. In *Joint Proceedings of SumPre 2015 and HSWI 2014-15*. CEUR-WS, 2015.
10. Sanda M. Harabagiu, Dan I. Moldovan, Marius Pasca, Rada Mihalcea, Mihai Surdeanu, Razvan C. Bunescu, Roxana Girju, Vasile Rus, and Paul Morarescu. FALCON: boosting knowledge for answer engines. In *TREC*, 2000.
11. Steve Harris and Andy Seaborne. SPARQL 1.1 Query language. W3C Recommendation, 21 March 2013.
12. Philipp Heim, Thomas Ertl, and Jürgen Ziegler. Facet Graphs: Complex Semantic Querying Made Easy. In *ESWC*, pages 288–302, 2010.

13. Hai Huang, Chengfei Liu, and Xiaofang Zhou. Computing Relaxed Answers on RDF Databases. In *WISE*, volume 5175 of *LNCS*, pages 163–175. Springer, 2008.
14. Evgeny Kharlamov, Luca Giacomelli, Evgeny Sherkhonov, Bernardo Cuenca Grau, Egor V. Kostylev, and Ian Horrocks. Semfacet: Making hard faceted search easier. In *CIKM*, pages 2475–2478, 2017.
15. Evgeny Kharlamov, Dag Hovland, Martin G. Skjæveland, Dimitris Bilidas, Ernesto Jiménez-Ruiz, Guohui Xiao, Ahmet Soylu, Davide Lanti, Martin Rezk, Dmitriy Zheleznyakov, Martin Giese, Hallstein Lie, Yannis E. Ioannidis, Yannis Kotidis, Manolis Koubarakis, and Arild Waaler. Ontology based data access in statoil. *J. Web Sem.*, 44:3–36, 2017.
16. Evgeny Kharlamov, Theofilos Mailis, Gulnar Mehdi, Christian Neuenstadt, Özgür L. Özçep, Mikhail Roshchin, Nina Solomakhina, Ahmet Soylu, Christoforos Svingos, Sebastian Brandt, Martin Giese, Yannis E. Ioannidis, Steffen Lamparter, Ralf Möller, Yannis Kotidis, and Arild Waaler. Semantic access to streaming and static data at siemens. *J. Web Sem.*, 44:54–74, 2017.
17. Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *AAAI*, pages 129–137, 2014.
18. Héctor Pérez-Urbina, Edgar Rodríguez-Díaz, Michael Grove, George Konstantinidis, and Evren Sirin. Evaluation of Query Rewriting Approaches for OWL 2. In *Proc. of SSWS+HPCSW*, 2012.
19. Alistair Russell and Paul R. Smart. NITELIGHT: A graphical editor for SPARQL queries. In *ISWC (Posters and Demos)*, 2008.
20. Ahmet Soylu, Martin Giese, Ernesto Jimenez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, and Ian Horrocks. Ontology-based end-user visual query formulation: Why, what, who, how, and which? *Universal Access in the Information Society*, 16(2):435–467, 2017.
21. Ahmet Soylu, Martin Giese, Ernesto Jimenez-Ruiz, Guillermo Vega-Gorgojo, and Ian Horrocks. Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. *Universal Access in the Information Society*, 15(1):129–152, 2016.
22. Ahmet Soylu, Evgeny Kharlamov, Dimitry Zheleznyakov, Ernesto Jimenez Ruiz, Martin Giese, Martin G. Skjaeveland, Dag Hovland, Rudolf Schlatte, Sebastian Brandt, Hallstein Lie, and Ian Horrocks. OptiqueVQS: a visual query system over ontologies for industry. *Semantic Web*, 9(5):627–660, 2018.
23. Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.
24. Daniel Tunkelang. *Faceted Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009.
25. Andreas Wagner, Günter Ladwig, and Thanh Tran. Browsing-oriented Semantic Faceted Search. In *DEXA*, pages 303–319, 2011.
26. Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. SPARK: adapting keyword query to semantic search. In *ISWC*, pages 694–707, 2007.