Towards Exploiting Query History for Adaptive Ontology-based Visual Query Formulation

Ahmet Soylu[1], Martin Giese[1], Ernesto Jimenez-Ruiz[2], Evgeny Kharlamov[2], Dmitriy Zheleznyakov[2], and Ian Horrocks[2]

[1] Department of Informatics, University of Oslo, Norway
{ahmets, martingi}@ifi.uio.no
[2] Department of Computer Science, University of Oxford, United Kingdom
{name.surname}@cs.ox.ac.uk

Abstract. Grounded on real industrial use cases, we recently proposed an ontology-based visual query system for SPARQL, named OptiqueVQS. Ontology-based visual query systems employ ontologies and visual representations to depict the domain of interest and queries, and are promising to enable end users without any technical background to access data on their own. However, even with considerably small ontologies, the number of ontology elements to choose from increases drastically, and hence hinders usability. Therefore, in this paper, we propose a method using the log of past queries for ranking and suggesting query extensions as a user types a query, and identify emerging issues to be addressed.

Keywords: Visual Query Formulation, Ontology-based Data Access, SPARQL, Ranking, Recommendation.

1 Introduction

In data-intensive organisations, *domain experts* usually meet their *information needs* either by operating a set of predefined *queries* embedded into applications or by involving *IT experts* to translate their information needs into queries. This is because domain experts often lack necessary *technical knowledge* and *skills* pertaining to query languages and databases. This man-in-the-middle approach for extracting data introduces a bottleneck in *data access* and consequently delays in *value creation* processes (cf. [1]).

Visual query formulation (cf. [2]) is a longstanding research endeavour and, though oriented towards a wide spectrum of users, a particularly prominent approach to mitigate the data access problem of users without any technical skills (i.e., *end users* – cf. [3]). This is due to fact that visual query formulation tools rely on *recognition*, rather than *recall*, and *direct manipulation* of objects, rather than a *command language syntax*, by using *visual representations* to depict the domain of interest and queries. In this context, we have recently introduced an ontology-based *visual query system* (*VQS*) for end users, named *OptiqueVQS* [4,5]. It is built on a scalable data access platform for *Big Data* developed within an EU

project called *Optique*[1] [1]. *Ontologies* provide reasoning support and a domain representation closer to end users' understanding, compared to earlier approaches built on low-level domain models (e.g., relational schemas) (cf. [6,2]). Besides, Optique employs an *ontology-based data access* (*OBDA*) (cf. [7,8]) technology that extends the platform's data access capabilities to traditional *relational data sources*, which store a significant amount of the world's enterprise data today.

One of the main problems that OptiqueVQS and typically any other VQS face is *scalability* against large ontologies (cf. [9]). A VQS has to provide its users with the elements of ontology (e.g., *concepts* and *properties*) continuously, so that users can select relevant ontology elements and iteratively construct their queries. However, even with considerably small ontologies, the number of concepts and properties to choose from increases drastically due to the propagative effect of ontological reasoning (cf. [10]). In turn, the high number of ontology elements overloads the user interface and hinders *usability*.

We approach the aforementioned problem with *adaptivity* (cf. [11]) by exploiting a *query history* to *rank* and *suggest* ontology elements with respect to an incomplete query that a user has constructed so far (i.e., *context-aware*). The approach is specifically devised for *SPARQL* [12], takes semantics into account with reasoning support, and uses SPARQL, as a programming language, for the implementation. In the rest of paper, we first describe OptiqueVQS, present our ranking proposal, and then discuss the related work. Finally, we provide a discussion on the proposal and emerging issues and conclude the paper.

2 OptiqueVQS

OptiqueVQS is built on multiple and coordinated *representation* and *interaction paradigms* (cf. [2]) and enables end users to formulate comparatively complex queries. OptiqueVQS has a *widget-based* architecture, which underpins its multi-paradigm approach and provides extensibility and flexibility. In the followings, we describe the interface and the formal aspects of SPARQL generation.

2.1 Interface

OptiqueVQS currently has three widgets, see Fig. 1: *W1* (see the top part of Fig. 1) employs a *diagram-based* representation paradigm, gives an overview of the constructed query, and allows further manipulation of it; *W2* (see the bottom-left part of Fig. 1) employs a *menu-based* representation paradigm along with *query by navigation* interaction style (cf. [13]) to let users join concepts via relationships connecting them; *W3* (see the bottom-right part of Fig. 1) is *form-based* and presents the attributes of a selected concept for *selection* and *projection* operations. W3 also has a *faceted search* flavour (cf. [14]), as it uses several natural interaction mechanisms, such as range sliders.

Query construction process in OptiqueVQS works as follows [4,5] – a demo is available[2]. The user begins with selecting a starting concept in W2, i.e., a

[1] `http://www.optique-project.eu/`
[2] `http://youtu.be/ks5tcPZVHp0`

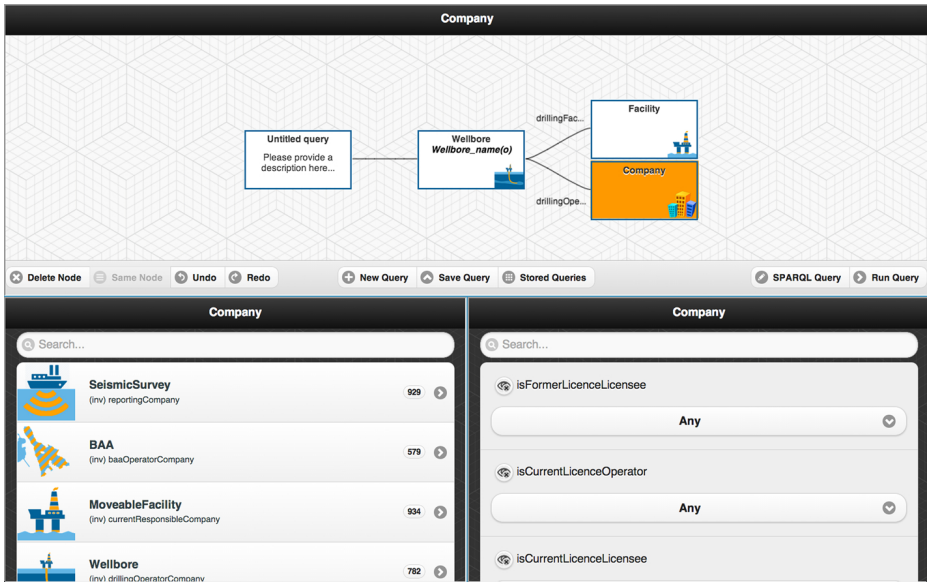

Fig. 1. OptiqueVQS – an example query is depicted.

kernel concept, the selected concept appears in W1 as a typed *variable-node*, and becomes *active* (aka *focus*, *pivot* etc.). Then, the user can extend the query either by selecting one of the offered concept-property pairs in W2, i.e., concepts reachable from the pivot via some *object property*, or by setting constraints on *data type properties* or selecting output variables in W3, i.e., by restricting the data properties of the objects belonging to the pivot. W3 also handles *subclass* selection, as it presents direct subclasses of the pivot concept as a multi-select form element. The user can change the pivot by clicking on any variable-node in W1 and continue extending the query by selecting a concept-property pair in W2. OptiqueVQS automatically extends the list of concept-property pairs and data properties in W2 and W3 via the *HermiT reasoner* [15] (e.g., a concept inherits all the properties of its parent concept). The user can delete nodes, access query catalogue, save/load queries, undo/redo actions, or continue query construction in the textual SPARQL mode.

2.2 Formal description

OptiqueVQS currently supports *linear* and *tree-shaped conjunctive* queries. The OBDA framework behind OptiqueVQS supports *OWL 2 QL* [16] and a conjunctive fragment of *SPARQL 1.1* [12]. OWL 2 QL is a *profile* of *OWL 2* and in this profile query answering can be implemented by rewriting queries into a standard relational query language [17].

The way the ontology controls the behaviour of OptiqueVQS should be seen from two perspectives: from a *knowledge representation* (*KR*) perspective, Optique exploits the graph-based organisation of ontological elements and data for representing the domain and query structures (cf. query by navigation); from a *logic* perspective, it uses ontological *axioms* to constrain the behaviour of the interface and to extend the available knowledge. On a purely structural level, OptiqueVQS could be controlled directly by a graph G that captures the concepts and the properties of an ontology O. An OWL ontology can be viewed as a *labeled directed* RDF graph $G = (N, E)$, where N is a finite set of labeled *nodes* and E is a finite set of labeled *edges* (cf. [17]). We consider pairwise disjoint alphabets U, a set of *URIs*, L, a set of *terminal literals*, and B, a set of *blank nodes*. An edge is a *triple* written in the form of $\langle s, p, o\rangle \in (U \cup B) \times U \times (U \cup L \cup B)$. The nodes of the graph mainly represent concepts and edges represent properties. A SPARQL query is formally represented by a tuple defined as $Q = (A, V, D, P, M, R)$. A is the set of *prefix declarations*, V is the *output form*, D is the *RDF graph* being queried, P is a *graph pattern*, M are *query modifiers*, which allow to modify the results by applying *projection*, *order*, *limit*, and *offset* options. SPARQL is based on matching graph patterns against RDF graphs. P is composed of a set of *triple patterns* and describes a *subgraph* of D. The main difference between a triple pattern and RDF triple comes from the fact that the former may have each of *subject*, *predicate* and *object* as a variable. However, once we substitute variables in triple patterns with constants or blank nodes, we reach an RDF graph $P'(N', E')$ that could be considered as a subgraph of the actual RDF data graph.

Every query generated by OptiqueVQS has a graph pattern represented by a set of triple patterns, where each triple pattern is a tuple $t \in Var \times U \times (U \cup Var \cup L)$ and Var is an infinite set of variables. The state of an edited query is composed of a partial graph pattern and a *cursor position* (cf. pivot). The cursor position is either blank (i.e., empty query) or points to a variable in the query. If the query is empty, the selection of a concept v from W2 results in a new tuple $\langle x, \texttt{rdf:type}, v\rangle \in Var \times U \times U$ in P, where x is a fresh variable. If the cursor points to a variable x, of type v, then each selection of a object property o with target class w from W1 (corresponding to an edge $\langle v, o, w\rangle \in G$) adds the following two triple patterns to P: $\langle x, o, y\rangle \in Var \times U \times Var$ and $\langle y, \texttt{rdf:type}, w\rangle \in Var \times U \times U$, where y is a fresh variable. Every selection and projection operation realised over a data property d in W3, while cursor is on a variable x, adds a new tuple $\langle x, d, y\rangle \in Var \times U \times (Var \cup L)$ to P. Finally, the selection of a subclass v for a typed variable x in W3 results in a new triple in P: $\langle x, \texttt{rdf:type}, v\rangle \in Var \times U \times U$.

3 Adaptive Query Formulation

Currently, the widgets W2 and W3 (see Fig. 1) present all the available concept-object property pairs and data properties to users respectively. However, the lists grow quickly due to *ontology size*, *number of relationships between concepts*,

subproperties, *inverse properties*, *inheritance of restrictions* etc. As the lists grow, the time required for a user to find elements of interest increases; therefore ranking ontology elements with respect to previously executed queries and suggesting highly ranked elements first as possible query continuations have potential to increase the efficiency of the users. The nature of OptiqueVQS requires suggestions to be done for the pivot (i.e., cursor point) rather than for any part of a query.

In what follows, we first present a running example and then describe our ranking method for context-aware suggestions. The running example is built on one of the industrial Optique use cases, namely the *Statoil* use case. Statoil[3] is a large international energy company focused on upstream oil and gas operations. The company reports that value creation processes could be improved considerably, if domain experts are to be able to access data on their own.

3.1 Running Example

The exploration department of Statoil has to find new hydrocarbon reserves in a cost effective way and ultimately the only way to prove the presence of a reserve is to drill an exploration well, which may consist of one or several well paths, i.e., wellbores. But since drilling is very expensive, it is important to maximise the chances of success. To do this, all available data from previous and ongoing exploration and production projects to extrapolate a model of the geology of a field, which then allows to anticipate the presence of hydrocarbon reserves.

A partial simplified ontology for Statoil exploration department is depicted in Fig. 2. The ontology currently contains 344 concepts, 148 object properties, 237 data properties, and 8190 axioms and it is yet to grow. In Fig. 3, an example query log with three queries is assumed for the sake of brevity. The first query, *Q1*, is the one that is depicted in Fig. 1 and asks for the names of wellbores with a drilling facility and a drilling company. The second query, *Q2*, asks for the content of all shallow wellbores that belongs to a well and has a drilling company of type operator. The final query, *Q3*, asks for the content of all exploration wellbores that has a fixed drilling facility and a drilling company.

In Fig. 3, *PQ* refers to an example partial query. The query in its incomplete form asks for all exploration wellbores with a drilling company; the cursor point is the variable of type exploration wellbore. At this point of query formulation session, the widgets W2 and W3 need to suggest the most relevant continuations, by comparing the partial query with the queries in the query log.

3.2 Ranking Method

A query log QL is basically a set of SPARQL queries: $QL = \{Q_1, Q_2, ..., Q_n\}$. We define a function p that takes a query Q as an input and returns its graph pattern P. We define S as a set suggestions $\{T_1, T_2, ..., T_m\}$. Each suggestion in S is a triple set T_i, which either contains two triples for W2 in the form of $\{\langle x, o, y\rangle \in Var \times U \times Var, \langle y, \texttt{rdf:type}, w\rangle \in Var \times U \times U\}$ or one triple for

[3] `http://www.statoil.com/`

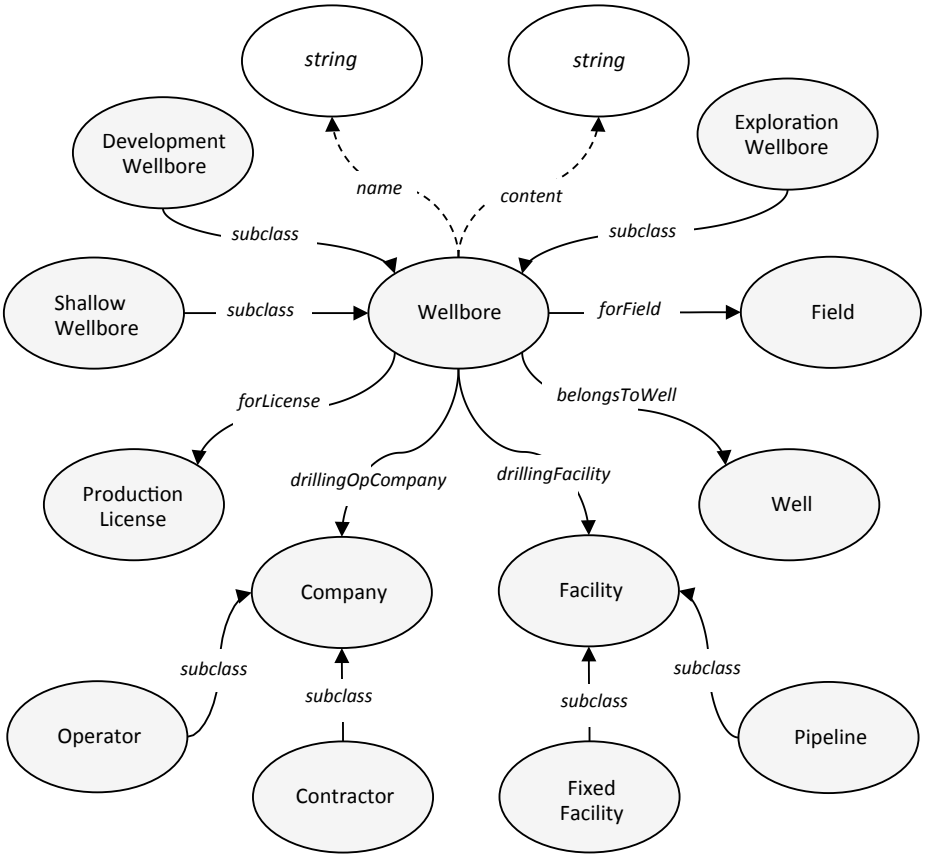

Fig. 2. A partial simplified ontology for the Statoil use case.

W3 in the form of $\{\langle x, d, y\rangle \in Var \times U \times (Var \cup L)\}$, where x corresponds to the cursor variable in a partial user query Q_a. Note that subclass suggestion is not included in the ranking, since it is always suggested by default.

The ranking score, at this point, basically corresponds to the *conditional probability* for each suggestion T_i in S, given a partial query Q_a and a query log QL, that is $Pr(T_i \mid p(Q_a))$. Conditional probability and probability functions are defined in the followings.

Within a query log QL, the probability of a graph pattern P is defined as the fraction of graph patterns in QL that are *supergraphs* [18] of P, as shown in Eq. 1.

$$Pr(P) = \frac{|\{Q_i \in QL | P \subseteq p(Q_i)\}|}{|QL|} \tag{1}$$

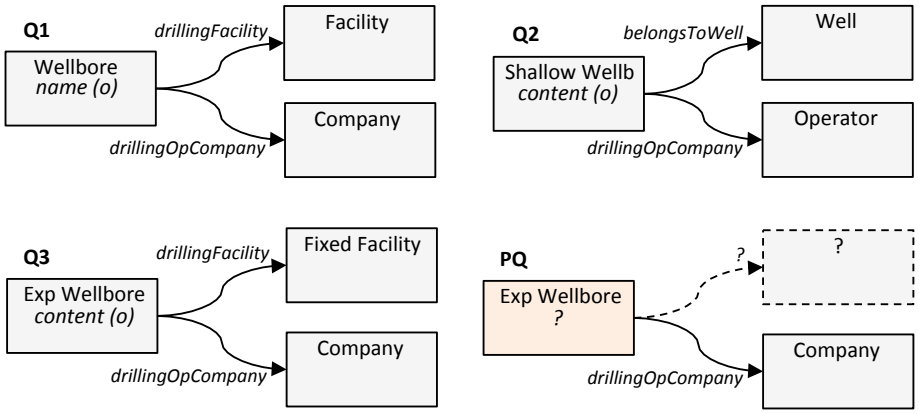

Fig. 3. A query log with three queries and an example partial user query.

The conditional probability of a triple set T given a graph pattern P is defined as the quotient of the probability of the union of T and P, and the probability of P as shown in Eq. 2.

$$Pr(T \mid P) = \frac{Pr(T \cap P)}{Pr(P)} \tag{2}$$

Now two important questions come into play. First, how do we find supergraphs in the query log, given a partial user query? Second, how do we extract possible extensions, i.e., suggestions, for the partial query from found supergraphs? As far as the first problem is concerned, it boils down to a *graph matching* problem. We consider a graph pattern P_1 to be subgraph of another graph pattern P_2, if all the triple patterns of P_1 are covered by P_2, independent of variable names, ordering of triple patterns, and the values of constraints. Dividino and Groner [19] review different approaches for checking graph similarity, where our interest falls into *content-based* approaches. We propose a method that relies on SPARQL itself and provides us with an exhaustive solution, as it allows us to exploit semantic knowledge while matching queries.

The method starts with the instantiation of graph patterns of queries in the query log by replacing variable names and constraints on data type properties with blank nodes; blank node names are marked with a query identifier for preventing any overlap and identification purposes. Then, the resulted RDF graphs are stored in a common dedicated triple store; the instantiation of the query log depicted in Fig. 3 is given in Fig. 4. By applying the partial query over this triple store, one can retrieve all the queries that are the supergraphs of the partial query.

As far as the second question is concerned, i.e., finding possible extensions, the partial query is extended with a triple pattern from the cursor point to retrieve all extensions occurred in the matching supergraphs. The output of partial query

Query log: SPARQL form

```
Q1 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:Wellbore.
  ?c2 ns1:type ns2:Facility.
  ?c3 ns1:type ns2:Company.
  ?c1 ns2:drillingFacility ?c2.
  ?c1 ns2:drillingOpCompany ?c3.
  ?c1 ns2:name ?a1.
}

Q2 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:ShallowWellbore.
  ?c2 ns1:type ns2:Operator.
  ?c3 ns1:type ns2:Well.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ns2:belongsToWell ?c3.
  ?c1 ns2:wellboreContent ?a2.
}

Q3 SELECT DISTINCT ?c1 ?a1 ?c2 ?c3
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c3 ns1:type ns2:FixedFacility.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ns2:drillingFacility ?c3.
  ?c1 ns2:wellboreContent ?a1.
}
```

Query log: triple form

```
_:q1c1 ns1:type ns2:Wellbore.
_:q1c2 ns1:type ns2:Facility.
_:q1c3 ns1:type ns2:Company.
_:q1c1 ns2:drillingFacility _:q1c2.
_:q1c1 ns2:drillingOpCompany _:q1c3.
_:q1c1 ns2:name _:q1a1.

_:q2c1 ns1:type ns2:ShallowWellbore.
_:q2c2 ns1:type ns2:Operator.
_:q2c3 ns1:type ns2:Well.
_:q2c1 ns2:drillingOpCompany _:q2c2.
_:q2c1 ns2:belongsToWell _:q2c3 .
_:q2c1 ns2:wellboreContent _:q2a1.

_:q3c1 ns1:type ns2:ExpWellbore.
_:q3c2 ns1:type ns2:Company.
_:q3c3 ns1:type ns2:FixedFacility.
_:q3c1 ns2:drillingOpCompany _:q3c2.
_:q3c1 ns2:drillingFacility _:q3c3.
_:q3c1 ns2:wellboreContent _:q3a1.
```

Fig. 4. The instantiation of query graph patterns.

is modified to retrieve the identifiers of matching queries, properties, and the types of variables for the returned extension. An example is given in Fig. 5 for the partial query depicted in Fig. 3 and the triple store depicted in Fig. 4. The rest of the method involves calculation of conditional probabilities for the suggestions, as exemplified in Fig. 5.

If one inspects the results in Fig. 5 closely, she will realise that reasoning is involved. This is because in the query log, only Q3 is an exact match for the partial query. However, thanks to reasoning support, Q1 is also matched, since exploration wellbore is a subclass of wellbore. Likewise, this guarantees a match for any query that has a *semantic similarity* [20] to the partial query, involving subclasses, subproperties, inverses etc. Later in the paper, this is to be discussed further, as there is a *semantic distance* involved between the partial query and the matched query. Yet, it is possible to query the triple store without any reasoning, if one wants to eliminate such matches, hence avoiding any semantic distance.

Partial user query

```
SELECT DISTINCT ?c1 ?c2
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c1 ns2:drillingOpCompany ?c2.
}
```

Modified partial user query

```
SELECT DISTINCT ?c3 ?prop ?type
WHERE {
  ?c1 ns1:type ns2:ExpWellbore.
  ?c2 ns1:type ns2:Company.
  ?c1 ns2:drillingOpCompany ?c2.
  ?c1 ?prop ?c3.
  OPTIONAL { ?c3 rdf:type ?type }
}
```

Matches

?c3		?prop	?type	Pr(T|P)	Widget
_:q1c2	T_1	ns2:drillingFacility	ns2:Facility	0.16	W2
_:q1c3	T_2	ns2:drillingOpCompany	ns2:Company	0.33	W2
_:q1a1	T_3	ns2:name		0.16	W3
_:q3c2	T_2	ns2:drillingOpCompany	ns2:Company		
_:q3c3	T_4	ns2:drillingFacility	ns2:FixedFacility	0.16	W2
_:q3a1	T_5	ns2:wellboreContent		0.16	W3

Fig. 5. Modified partial user query and possible query extensions.

The final stage involves ordering and dividing S into two sets, S_1 for W2 and S_2 for W3, with respect to ranking score and type of each suggestion (i.e., concept-relationship pair vs. data type property). Then, suggestions in each set are paginated into $\frac{|S_i|}{j}$ pages, where i is the set identifier and j is the *window size* for a page (i.e., the required number of suggestions for a page).

4 Related Work

There are a number of visual query formulation tools available in the literature (e.g., [2,21,22,23]); however, to the best of authors knowledge none of them supports adaptive visual query formulation. Existing approaches for adaptive query formulation are largely developed for *context-sensitive* textual query formulation.

Khoussainova et al. [24] provide a system, named *SnipSuggest*, for context-aware composition of textual SQL queries with respect to a given query log. The authors translate each SQL query in the query log into a set of features (e.g., a table name appearing in the *FROM clause*). Similarly, the partial query of the user is also translated into a set of features. Possible features for extension are identified by matching the feature sets of the partial query and the feature sets of queries in the query log and are ranked by calculating conditional probabilities. The approach generates suggestions for extending any part of the partial query rather

than a single cursor point. Authors also propose a set of supportive algorithms and techniques for, such as feature set matching (i.e., what if the partial query does not appear in the query log), the selection of suggestions (i.e., accuracy vs. diversity), and query log elimination (i.e., to reduce the size). The elaborate approach provided by SnipSuggest system is relevant to our contribution in many aspects. However, a fundamental difference is in feature comparison; while the features of SnipSuggest system are a set of syntactic elements and the feature comparison is string based, for OptiqueVQS feature sets (i.e., correspond to the triple sets of graph patterns) have a semantic nature and compared semantically. The semantic aspects not only concern how the matching is done, but also the calculation of rankings, which we discuss in the following section.

As far as approaches for SPARQL are concerned, Campinas et al. [25] propose an approach for assisting textual SPARQL query formulation, however in a different context. The approach assumes that an ontology describing the data set is unknown. Therefore, the authors propose a model that summarises the underlying data graph and extracts ontology elements to suggest. The approach extends a given partial user query from the cursor point, similar to our approach, and then evaluates it over the data graph summary to retrieve possible extensions. However, the approach does not realise any ranking of suggestions based on the previously executed queries and does not take semantic similarities between queries into account, possibly due to lack of rich domain knowledge (e.g., lack of subclass, inverse property axioms).

Kramer et al. [26] present a tool, named *SPACE*, to support autocompletion of textual SPARQL queries. For this purpose, it takes a SPARQL query log as an input and then builds an *index structure* for the computation of query suggestions. The index structure has a root node at level 0, representing a set of queries, while each vertex at level 1 represents a SPARQL query. The vertices from level $n-2$ to index level 1 represent graph patterns recursively. Finally the vertices at the highest level $(n-1)$ represent IRIs, blank nodes, literals, variables, and binary operators such as *AND*, *UNION*, and *FILTER*. The suggestion process is done by subgraph matching for the partial user query in the index graph in a bottom up manner. However, the authors describe neither the subgraph matching process nor the details of ranking calculation. Finally, the index structure could grow quickly as it is built on recursive decomposition of graph patterns.

5 Discussion

The fact that there exist SPARQL engines capable of handling large triple sets effectively [27,28] is a positive evidence for the execution performance of our approach, since our proposal relies on SPARQL querying for matching partial user queries against a query log. One should also note that the size of a triple store for a query log is only expected to be in the order of thousands triples, if maintained – e.g., pruned, clustered etc.

As far as the *precision* of suggestions is concerned, approaches that take the partial query into account are reported to be better than *popularity-based*

approaches purely built on the number of occurrences of terms in the query log [24]. Note that, initially, when no kernel concept is selected, our approach behaves like a popularity-based approach, as it extends an empty query. Below, we discuss a set of issues that need to be addressed:

Semantic distance: In Fig. 3, the match between the first query and partial query is due to their semantic similarity and is not exact (e.g., exploration wellbore is a subclass of wellbore); and in Fig. 5, the drillingFacility - Facility and drillingFacility - FixedFacility suggestions are semantically similar. Therefore one could incorporate the semantic distance involved as a cofactor into the ranking function, so that semantically distant queries contribute less to the ranking. Huang et al. [20] suggest a similarity measure, which can readily incorporated to our proposal. It uses 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 combine them to compute similarity between triple patterns, hence queries.

No match: A problematic situation arises when no match is found for the partial query in the query log (cf. [24]). A possible solution could be pruning the partial query until a match is found. At each step of a pruning process, a leaf node, which is not the cursor point, could be randomly selected and deleted (or with respect to some heuristics), so that partial query graph pattern does not get disconnected and the cursor point is preserved.

Cold start: The proposal cannot draw any suggestions, when the query log contains no or insufficient number of queries. Mostly likely sources to use for addressing this problem are the ontology and data set. A statistical inspection of ontology, e.g., concept centrality with respect to the number of incoming and outgoing relationships, and the data set, e.g., the number of times each concept and property appears in the dataset, could reveal useful information to overcome the cold start problem.

Collective, group, or individual: The ranking and suggestions could be applied on an individual basis for each user, i.e., only over the portion of query log that belongs to the subject user, on group basis, i.e., only over the portion of query log that belongs to the users of same type, and on a collective basis, i.e., over the whole query log for every user (cf. [11]). The decision possibly should consider whether users are homogeneous or there exist different user groups, each using a part of the ontology heavily – e.g., geologist and chemists. In the former case, a group or even user specific approach is more feasible, as each user group/user focuses on a specific part of the ontology.

6 Conclusion and Future Work

Ontology-based end-user visual query formulation is promising for enhancing value creation processes; yet existing approaches are not scalable against large ontologies. Although there are some attempts for assisted textual query formulation in the literature; they are either not elaborate enough to be readily used in our case or do not take previously executed queries into account. In this paper, we proposed

a method for ranking and suggesting SPARQL query extensions, which relies on the partial user query, the queries in the query history, and their semantic similarity. We also identified notable issues to be addressed in order to reach an elaborate solution.

The future work involves comparative evaluation of the proposed method and its variants (e.g., with/without semantic similarity) in terms of precision. End-user studies are also planned to measure the *perceived usefulness*, i.e., whether in practice users find ranking approach useful or not.

Acknowledgements. This research is funded by the FP7 of the European Commission under Grant Agreement 318338, "Optique".

References

1. Giese, M., Calvanese, D., Horrocks, I., Ioannidis, Y., Klappi, H., Koubarakis, M., Lenzerini, M., Moller, R., Ozcep, O., Rodriguez Muro, M., Rosati, R., Schlatte, R., Soylu, A., Waaler, A.: Scalable End-user Access to Big Data. In Rajendra, A., ed.: Big Data Computing. Chapman and Hall/CRC (2013)
2. Catarci, T., Costabile, M.F., Levialdi, S., Batini, C.: Visual query systems for databases: A survey. Journal of Visual Languages and Computing **8**(2) (1997) 215–260
3. Lieberman, H., Paternó, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm. In Lieberman, H., Paternó, F., Wulf, V., eds.: End-User Development. Volume 9 of Human-Computer Interaction Series. Springer, Netherlands (2006) 1–8
4. Soylu, A., Giese, M., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I.: OptiqueVQS – Towards an Ontology-based Visual Query System for Big Data. In: Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES 2013), ACM (2013) 119–126
5. Soylu, A., Skjæveland, M., Giese, M., Horrocks, I., Jimenez-Ruiz, E., Kharlamov, E., Zheleznyakov, D.: A Preliminary Approach on Ontology-based Visual Query Formulation for Big Data. In: Proceedings of the 7th International Conference on Metadata and Semantic Research (MTSR 2013). Volume 390 of CCIS., Springer (2013) 201–212
6. Siau, K.L., Chan, H.C., Wei, K.K.: Effects of query complexity and learning on novice user query performance with conceptual and logical database interfaces. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans **34**(2) (2004) 276–281
7. Spanos, D.E., Stavrou, P., Mitrou, N.: Bringing relational databases into the Semantic Web: A survey. Semantic Web **3**(2) (2012) 169–209
8. Kogalovsky, M.R.: Ontology-Based Data Access Systems. Programming and Computer Software **38**(4) (2012) 167–182
9. Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., Giannopoulou, E.: Ontology visualization methods - A survey. ACM Computing Surveys **39**(4) (2007) 10:1–10:43
10. Grau, B.C., Giese, M., Horrocks, I., Hubauer, T., Jimenez-Ruiz, E., Kharlamov, E., Schmidt, M., Soylu, A., Zheleznyakov, D.: Towards Query Formulation and Query-Driven Ontology Extensions in OBDA Systems. In: Proceedings of the 10th OWL: Experiences and Directions Workshop (OWLED 2013). Volume 1080 of CEUR Workshop Proceedings., CEUR-WS.org (2013)

11. Brusilovsky, P., Kobsa, A., Nejdl, W., eds.: The Adaptive Web: Methods and Strategies of Web Personalization. Volume 4321 of LNCS. Springer (2007)
12. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. W3C Recommendation, W3C (March 2013)
13. Ter Hofstede, A.H.M., Proper, H.A., Van Der Weide, T.P.: Query formulation as an information retrieval problem. Computer Journal **39**(4) (1996) 255–274
14. Tunkelang, D., Marchionini, G.: Faceted Search. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan and Claypool Publishers (2009)
15. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research **36**(1) (2009) 165–228
16. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles. W3C Recommendation, W3C (October 2009)
17. Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: The Next Step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web **6**(4) (2008) 309–322
18. Ray, S.S.: Subgraphs, Paths, and Connected Graphs. In: Graph Theory with Algorithms and its Applications. Springer India (2013)
19. Dividino, R., Groner, G.: Which of the following SPARQL Queries are Similar? Why? In: Proceedings of the 1st International Workshop on Linked Data for Information Extraction (LD4IE 2013). Volume 1057 of CEUR Workshop Proceedings., CEUR-WS.org (2013)
20. Huang, H., Liu, C., Zhou, X.: Computing Relaxed Answers on RDF Databases. In: Proceedings of the 9th International Conference on Web Information Systems Engineering (WISE 2008). Volume 5175 of LNCS., Springer (2008) 163–175
21. Catarci, T., Dongilli, P., Di Mascio, T., Franconi, E., Santucci, G., Tessaris, S.: An ontology based visual tool for query formulation support. In: Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004). Volume 110 of Frontiers in Artificial Intelligence and Applications., IOS Press (2004) 308–312
22. Kapetanios, E., Baer, D., Groenewoud, P.: Simplifying syntactic and semantic parsing of NL-based queries in advanced application domains. Data & Knowledge Engineering **55**(1) (2005) 38–58
23. Barzdins, G., Liepins, E., Veilande, M., Zviedris, M.: Ontology Enabled Graphical Database Query Tool for End-Users. In: Proceedings of the 8th International Baltic Conference on Databases and Information Systems (DB&IS 2008). Volume 187 of Frontiers in Artificial Intelligence and Applications., IOS Press (2009) 105–116
24. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: SnipSuggest: Context-aware Autocompletion for SQL. Proceedings of the VLDB Endowment **4**(1) (2010) 22–33
25. 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 23rd International Workshop on Database and Expert Systems Applications (DEXA 2012), IEEE (2012) 261–266
26. Kramer, K., Dividino, R., Groner, G.: SPACE: SPARQL Index for Efficient Autocompletion. In: Proceedings of the ISWC 2013 Posters & Demonstrations Track (ISWC-PD 2013). Volume 1035 of CEUR Workshop Proceedings., CEUR-WS.org (2013)
27. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^2Bench: A SPARQL Performance Benchmark. In: Proceedings of the IEEE International Conference on Data Engineering (ICDE 2009), IEEE Computer Society (2009) 222–233
28. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems **5**(2) (2009) 1–24