


SPARQL Query Recommendation by Example: Assessing the Impact of Structural Analysis on Star-Shaped Queries

Alessandro Adamou 

Data Science Institute, National University of Ireland Galway, Ireland
alessandro.adamou@nuigalway.ie

Carlo Allocca 

Samsung Inc., London, United Kingdom
c.allocca@samsung.com

Mathieu d'Aquin 

Data Science Institute, National University of Ireland Galway, Ireland
mathieu.daquin@nuigalway.ie

Enrico Motta 

Knowledge Media Institute, The Open University, Milton Keynes, United Kingdom
enrico.motta@open.ac.uk

Abstract

One of the existing query recommendation strategies for unknown datasets is “by example”, i.e. based on a query that the user already knows how to formulate on another dataset within a similar domain. In this paper we measure what contribution a structural analysis of the query and the datasets can bring to a recommendation strategy, to go alongside approaches that provide a semantic analysis. Here we concentrate on the case of star-shaped SPARQL queries over RDF datasets.

The illustrated strategy performs a least general generalization on the given query, computes the specializations of it that are satisfiable by the target dataset, and organizes them into a graph. It then visits the graph to recommend first the reformulated queries that reflect the original query as closely as possible. This approach does not rely upon a semantic mapping between the two datasets. An implementation as part of the SQUIRE query recommendation library is discussed.

2012 ACM Subject Classification Information systems → Semantic web description languages

Keywords and phrases SPARQL, query recommendation, query structure, dataset profiling

Digital Object Identifier 10.4230/OASICS.LDK.2019.1

Category Short Paper

Acknowledgements This work was supported by the MK:Smart project (OU Ref. HGCK B4466).

1 Introduction

One of the main characteristics of the Linked Open Data (LOD) cloud is the heterogeneity of schemas and vocabularies: it is not rare for RDF datasets to use different vocabularies to describe similar domains. However, this also entails a proliferation of ontologies for overlapping domains appearing across datasets. For example, universities and academic institutions intermittently use AIISO¹ or XCRI² to describe their courses in RDF, and occasionally use the same properties in different ways. This increases the difficulty to build, for instance, a recommender system for courses offered by all the universities in

¹ Academic Institution Internal Structure Ontology, <http://purl.org/vocab/aiiso/schema#>

² eXchanging Course-Related Information, <http://xcric.org/>



© Alessandro Adamou, Carlo Allocca, Mathieu d'Aquin, and Enrico Motta;
licensed under Creative Commons License CC-BY

2nd Conference on Language, Data and Knowledge (LDK 2019).

Editors: Maria Eskevich, Gerard de Melo, Christian Fäth, John P. McCrae, Paul Buitelaar, Christian Chiarcos, Bettina Klimek, and Milan Dojchinovski; Article No. 1; pp. 1:1–1:8



OpenAccess Series in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a country. Finding the right queries for all the datasets usually involves intensive and time-consuming preliminary work to explore and understand each dataset’s data model and content, then iteratively reformulate and test SPARQL queries [13]. However, if the user has prior knowledge of some of the datasets and a tool that can exploit such knowledge to aid them in the reformulation, this effort can be reduced. Semantic analysis techniques such as ontology alignment provide support to querying unknown datasets [11], however most of them only consider how the terms are defined in the corresponding ontologies, hardly considering their roles in the datasets in terms of relationships between the structure of the query and that of the dataset. We investigate how much query recommendation “by example” can benefit from structural analysis for recommending the best queries soonest.

We propose a method that, given a SPARQL query q_o that a source RDF dataset D^S is able to answer, reformulates it into queries that can be answered by a target RDF dataset D^T and reflect as closely as possible the intended meaning, structure and types of results of the original query. The approach analyzes the structure of the query and relates it to the schema of both datasets, by computing a *least general generalization* and navigating the resulting operational structure of its specializations. It does not require an ontology mapping and/or instance matching between the datasets, but such methods can be integrated with it, by influencing the recommendation ranking. Basic constructs of RDF(S) and OWL are considered, but only as potential invariants in the query: in other words, the signature of a class is merely treated like a structure. In this paper, we concentrate on *star-shaped* queries, a class of queries that still represents a significant challenge in query optimization [8, 12].

The implementation of the method has been included with an open source SPARQL query recommendation toolkit called SQUIRE [1], which comes as a software library, standalone program and Web application. This offers valuable support for query recommendation, empowering not only the automatic learning of the data model and content of an RDF dataset, but also its straightforward use without the user’s prior knowledge of its content.

After illustrating related work in Section 2, the method is detailed in Section 3. Section 4 provides some insight on the experiments underway, before concluding with future work.

2 Related Work

Most research on SPARQL query recommendation uses ontology alignments or schema mappings explicitly defined between the source and target dataset [5, 10]. We are not aware of any study on how to recommend queries over unmapped datasets based not on a full schema, but on just enough knowledge of one to write example queries over another dataset.

Our work was partly inspired by the intuition that the capabilities of a dataset, in terms of what questions it is able to answer, can be understood by organizing them into a data structure that can be navigated through methods such as formal concept analysis [6].

The employment of least general generalizations is not new in semi-automated SPARQL querying: we acknowledge that Lehmann et al have previously implemented it with success for OWL-based machine learning in DL-Learner [3].

Although the above studies contribute interesting elements for us to build on, they were mainly driven either by the user’s lack of familiarity with the underlying technologies (which is not our case), or by the availability of semantic alignments. Regarding the latter, the placement of our work is immediately *before* semantic analysis takes place, in an effort to understand if such techniques can be further led to converge towards ideal recommendations.

Lastly we acknowledge the existence of effective methods, such as that by Fokou et al [7], to relax failing queries, rather than outright avoid them, for obtaining satisfiable ones. We are in fact in contact with the authors to investigate a possible intertwining of both techniques.

3 Method

A *star-shaped* SPARQL query is a query such that all its graph pattern expressions (GPEs) are either triple patterns (*subject, predicate, object*) that share the same subject, or GPEs obtained by combining them using the AND operator. For instance, the following:

```
SELECT DISTINCT ?title ?pic
WHERE {
  ?s rdf:type bibo:Book ; dc:title ?title ; foaf:depiction ?pic .
}
```

is a star-shaped query of 3 TPs to get the titles and pictures (e.g. front covers) of books.

To generate query recommendations by example, we first (i) deconstruct the original query into one or more general queries satisfiable by both datasets (*Generalization*); (ii) transform the general queries into queries for the target dataset (*Specialization*); and (iii) grade and rank the generated queries to select which ones to recommend (*Evaluation*).

3.1 Generalization Step

Suppose a query q_o is satisfiable (i.e. produces a non-empty result set) w.r.t. a dataset D^S but not necessarily D^T . The generalization step produces a set of queries $\mathcal{Q}^G = \{q_i^G, i = 1..m\}$ so that every q_i^G is satisfiable by both D^S and D^T , but if a non-redundant and non-trivial triple pattern (i.e. one that is not composed solely of variables and is not a repetition of an existing triple pattern in q_i^G) were added to it, the resulting query would no longer be satisfiable by both datasets. This is called a “least general generalization” (lgg) of q_o .

When a TP cannot be preserved in an lgg, for it would make the query no longer satisfiable by both datasets, its terms are excluded from the generalization and replaced with special, unique SPARQL variables called *template variables*. These variables use the convention ct_j , opt_j and dpt_j , respectively for the j -th class, object property and data property template variable. The sets of terms of each category for D^S and D^T are obtained by inspecting each dataset and maintaining an index of the terms. The ppt_j convention is used for “plain” (RDF) properties when it cannot be determined if they are being *used as* object or datatype properties. To reduce the risk of combinatorial explosion of recommendations, named individuals and literals in a TP are not replaced with template variables. It follows by construction that the members of an lgg are themselves queries.

Algorithm 1 below illustrates this rationale. Given the initial query q_o (2-5): take every class or property that appears in D^S but not in D^T and substitute every occurrence of it with an occurrence of a new template variable. At this point (6) we have one generalized query, but it is not guaranteed to be satisfiable by D^T . Therefore, build a property co-occurrence matrix M (7), which is a square matrix indexed by properties in D^T that indicates the classes, if any, where they appear together. If there are still concrete classes left, (8-12) add to \mathcal{Q}^G one query for each of them and generalize in each query every property that does not occur (i.e. does not co-occur with itself in M) for members of that class. Now (13) take the queries that were generated in the steps before and for each (14-17) find in M the largest sets of co-occurring properties appearing in that query (P_j is the set of their indices), then add to \mathcal{Q}^G one query for every such group, so that only properties of that group appear in it. If steps 7-17 did not produce any queries, then the query produced before was already general enough, therefore (18) make that the lgg.

Algorithm 1 Least general generalization of an input query.

INPUT: two datasets D^S and D^T ; a query q_o that is satisfiable w.r.t. D^S

OUTPUT: a set \mathcal{Q}^G of queries that generalize q_o and are satisfiable w.r.t. D^S and D^T

```

1: procedure GENERALIZE( $q_o, D^S, D^T$ )
2:   for each non-variable node  $p_i$  of  $q_o$  do
3:     if  $p_i$  is an object property in  $D^S$  but not in  $D^T$  then
4:       Replace every occurrence of  $p_i$  with a new object property template variable.
5:       Do the same for data properties, other properties and classes of  $D^S$ .
6:    $q_o^G \leftarrow$  the new query resulting from the above
7:    $M \leftarrow$  co-occurrence matrix of all the properties in  $D^T$  w.r.t classes in  $D^T$ 
8:   for each concrete class  $c_i$  in  $q_o^G$  do
9:     Generate a new query  $q_{c_i}^G$  with all the TPs of  $q_o^G$  where  $c_i$  occurs
10:    for each predicate  $p_j$  of  $q_o^G$  that is not rdf:type do
11:      if  $c_i \in M_{jj}$  then add to  $q_{c_i}^G$  all the TPs with  $p_j$ 
12:      else add to  $q_{c_i}^G$  all the corresponding TPs after generalizing  $p_j$ 
13:    $\mathcal{Q}^G \leftarrow \{q_{c_i}^G\}_i$ 
14:   for each  $q_i^G \in \mathcal{Q}^G$  if  $\mathcal{Q}^G \neq \emptyset$  otherwise  $q_o^G$  do
15:     for each  $p_j$  occurring in  $q_i^G$  do
16:        $P_j \leftarrow$  largest group of properties  $p_k$  so that  $M_{jk} \neq \emptyset$ 
17:       Add to  $\mathcal{Q}^G$  a new query with only the TPs of  $q_i^G$  whose predicates are in  $P_j$ 
18:   if  $\mathcal{Q}^G = \emptyset$  then  $\mathcal{Q}^G \leftarrow \{q_o^G\}$ 

```

For example, suppose the first round of generalization has produced a query:

```

SELECT DISTINCT ?title ?pic WHERE {
  ?s rdf:type ?ct1 ; dc:title ?title ; foaf:depiction ?pic .
}

```

and `dc:title` and `foaf:depiction` are both present in D^T but never together. In that case, the following rounds will generate two queries (the largest property groups being $\{\text{rdf:type}, \text{dc:title}\}$ and $\{\text{rdf:type}, \text{foaf:depiction}\}$), whose patterns are respectively:

```
?s rdf:type ?ct1 ; ?dpt1 ?title ; foaf:depiction ?pic .
```

```
?s rdf:type ?ct1 ; dc:title ?title ; ?opt1 ?pic .
```

Having computed the set that represents the lgg of the original query, we need to know in which ways it can be transformed into a set of queries over the target dataset D^T , and detect those queries that are potentially closer to the original one. This is what specialization does.

3.2 Specialization Step

The goal of specializing an lgg \mathcal{Q}^G is to generate the space of candidate queries, regardless of which query is to be preferred over which. A necessary condition for a query to be among the candidates is that it must be satisfiable by the target dataset D^T .

Specialization can be regarded as the repeated application of some operation over a query. We distinguish two such operations: *Removal* (*Rop*) and *Instantiation* (*Iop*).

Removal (Rop) systematically removes an entire TP from a query, as well as removing from its projection (e.g. the SELECT statement) any variable that appeared only in that TP. Along with the obvious precondition of there being more than one TP in the query for *Rop* to be applied, we also restrict to only applying *Rop* on TPs that contain at least one template variable.

Instantiation (Iop) replaces every occurrence of a template variable in a query with one concrete (non-variable) value, thereby instantiating the template variable in question.

Applied to our specialization method, removals may be performed on a query regardless of the target dataset, since by monotonicity no solutions are lost if a TP is removed from an intersection of TPs. Instantiations, on the other hand, are performed so as to preserve the satisfiability of the query. To do so, *Iop* only replaces template variables with concrete values that occur in the target dataset. This requires knowledge of what the applicable instantiations are which, when repeatedly applied to a generalized query, produce a query that is satisfiable by the target dataset. One practical way to know them is to query the target dataset for them. We therefore take each query in Q^G and replace all the variables in the projection with all the template variables. So if a generalized query is:

```
SELECT DISTINCT ?s ?title ?pic WHERE {
  ?s rdf:type ?ct1 ; dc:title ?title ; ?opt1 ?pic .
}
```

then the discovery query for possible instantiations is the same as the one above, except that the projection, i.e. the variables in the SELECT clause, becomes `?ct1 ?opt1`.

The resulting query is run through D^T . Every solution returned (the URI bindings for `ct1`, `opt1`) denotes the possible instantiations that generate a candidate query for recommendation.

The set of solutions for every such query derived from Q^G defines the space of candidate queries for recommendation that can be obtained through instantiation.

Every time an operation from the specialization step is applied, a new query is generated, which reduces the occurrences of template variables compared to the query from before the operation was applied. Any query generated in this way is an *intermediate query* if at least one template variable occurs, or a *candidate query* otherwise. Generalized, intermediate and candidate queries are organized in a structure that is explored in the next step. To that end, create a directed graph (digraph) $g = (V_g, E_g)$, called **specialization graph**, so that:

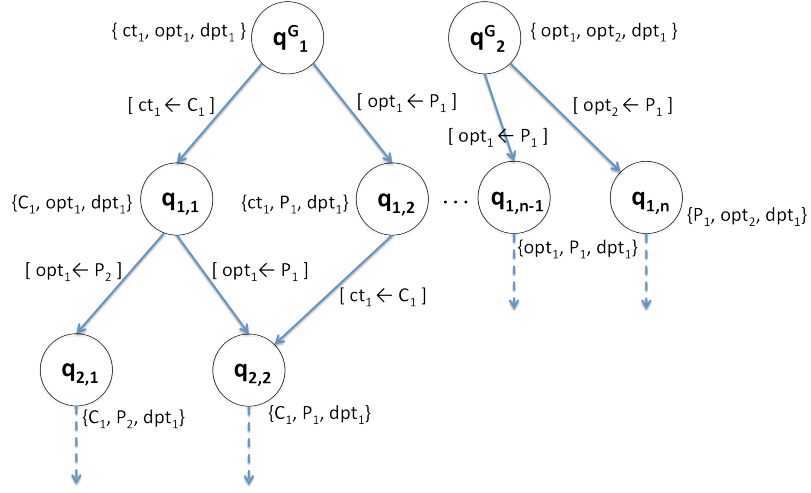
1. Every generalized query q_{0l} in Q^G is a vertex in V_g .
2. If a vertex q_{ij} is in V_g and an operation in the specialization step can be applied to it, so that a new query q_{i+1k} is generated, then $q_{i+1k} \in V_g$ and $(q_{ij}, q_{i+1k}) \in E_g$.

It follows that g contains all the generalized, intermediate and candidate queries as vertices, and that the candidate queries have no outgoing edges. Also every applicable operation on a query denotes an edge in g labelled after it. Figure 1 shows an example.

The next step is to navigate the specialization graph in order to pick the best candidate queries, trying to detect them as early as possible.

3.3 Evaluation Step

The goal now is to measure the appropriateness of a candidate query using distance measures from one of the generalized queries. With the specialization graph in place, if we set weights on its edges we will have reduced our problem to one of finding the shortest paths to traverse a digraph, from a generalized query to a candidate query. Shortest path traversal can be performed using textbook algorithms that compute the least costly paths. However, in order to assign weights to the edges, we will need to have a cost model.



■ **Figure 1** Example specialization graph of an lgg $Q^G = \{q_1^G, q_2^G\}$ (unweighted). The edges are annotated with the last operation performed, whereas the vertices are annotated with the sets of (remaining) template variables and/or the values used to substitute them in the corresponding query.

As there is no unique way of measuring the distance and/or similarity between two queries [9], the cost C of going from q_i to a q_j satisfiable w.r.t. D is a linear combination:

$$C(q_i, q_j, D) = \alpha \cdot qsd(q_i, q_j) + \beta \cdot tpc(q_i, q_j) + \gamma \cdot ppd(q_i, q_j, D) + \delta \cdot \Phi \quad (1)$$

where $\alpha, \beta, \gamma, \delta$ are arbitrarily set coefficients that depend on how we wish to reward or penalize a certain measure, and qsd , tpc and ppd are measures defined as follows:

Query Specificity Distance (qsd) measures the alterations of variables and triple patterns:

$$qsd(q_i, q_j) = qsd^{Var}(q_i, q_j) + qsd^{TP}(q_i, q_j) \quad (2)$$

qsd^{Var} is the ratio of variables in q_i that were preserved by the operation, over those of q_i and q_j combined. qsd^{TP} does the same with the triple patterns of q_i that were preserved.

Triple Pattern Collapse (tpc) measures the *increase* in occurrences of every concrete value from one query to another. If $occ(x, q)$ extracts the set of occurrences (*triplepattern, role*) of a variable or value x in a query q , then:

$$tpc(q_i, q_j) = \sum_{u \in concrete(q_i)} ||occ(u, q_j) \setminus occ(u, q_i)|| \quad (3)$$

Only Iop operations can cause an increase in value occurrence, by instantiating one or more occurrences of a template variable to an existing URI value. The side effect is that two or more TPs are *collapsed* by rendering them virtually indistinguishable. This measure imposes a penalty on queries with such TPs. Rop operations are not penalized.

Property Preservation Distance (ppd) counts the properties whose nature has changed across datasets, e.g. replacing an object property with a datatype property or vice versa.

Finally, Φ is a black-box similarity measure, which can be based on the syntactic or semantic analysis of the query. It is treated like a black box here, since we assume to know nothing about it: at a minimum one could use simple string similarity, just to allow us to prefer one edge over others that would otherwise be equally weighted.

We apply the cost function $C(q_{ij}, q_{i+1k}, D^T)$ to all the vertices connected by an edge and use the result as the weight of that edge. We can then proceed to visit the graph. The least costly paths, from a vertex that represents a generalized query, to each final vertex, can be found using an algorithm such as Bellmann-Ford or Dijkstra, depending on whether the coefficients $[\alpha - \delta]$ can be negative or not [2]. The resulting list of final vertices, already sorted by ascending cost to reach, constitutes the query recommendation.

4 Implementation

Our approach was implemented as part of the SQUIRE open source toolkit for SPARQL query recommendation³. SQUIRE provides query recommendation facilities packaged as: (a) a Java library to be included into other programs; (b) a standalone application from the command line; (c) a Web Service with an associated Web Application frontend. SQUIRE supports Lucene-based⁴ dataset indexing to inform the *generalization* and *specialization* steps of the method. The indices were populated through paginated exploratory SPARQL queries.

The base implementation of SQUIRE can be extended with metrics based, for instance, on the semantic analysis of queries and datasets (i.e. the Φ in Formula (1)). As we are currently evaluating our approach, we are minimizing the bias of this factor by instead using an intentionally naïve Jaro-Winkler similarity [4] computed on the class and property labels, or on the path ends or fragments of their URIs. We are constructing a suite of queries a datasets pairs to benchmark query recommendation. At the time of writing, the optimal recommendation could be found among the first five candidates for most queries tried so far.

5 Conclusions and future directions

We have illustrated an approach to the recommendation of SPARQL queries for datasets that the user does not know, based on a query that they are already able to formulate for another dataset. Our initial study concentrated on assessing how far it is possible to go with an analysis of the query and datasets that is largely structural, only considering basic RDFS and OWL constructs, such as classes as properties, as potential invariants of such structures. We started with star-shaped queries, an essential yet already challenging structural category.

With experimental validation currently underway, we noted that there is no benchmark for SPARQL recommendation, therefore we set out to publish one to complement our evaluation. As for extending the method itself, there are several directions to take. One is to extend the structural analysis to other classes of queries, such as *snowflake* and *chain*. Another one is of course to introduce other types of analysis (such as those based on semantic alignments or subsumption hierarchy) and measure if our structural analysis aids the convergence to better recommendations sooner than without it. Having multiple analysis techniques will also allow a comparative evaluation of the method's efficiency. Finally, we will keep refining the structural analysis method itself, especially considering other types of operation for generalizing and specializing queries, and possibly taking FILTER clauses into account.

³ SQUIRE on GitHub, <https://github.com/carloallocca/Squire>

⁴ Apache Lucene, <http://lucene.apache.org>

References

- 1 Carlo Allocca, Alessandro Adamou, Mathieu d'Aquin, and Enrico Motta. SPARQL query recommendations by example. In Harald Sack, Giuseppe Rizzo, Nadine Steinmetz, Dunja Mladenic, Sören Auer, and Christoph Lange, editors, *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers*, volume 9989 of *Lecture Notes in Computer Science*, pages 128–133, 2016. doi:10.1007/978-3-319-47602-5_26.
- 2 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs - theory, algorithms and applications*. Springer, 2002.
- 3 Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. DL-Learner - A framework for inductive learning on the Semantic Web. *J. Web Sem.*, 39:15–24, 2016. doi:10.1016/j.websem.2016.06.001.
- 4 William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 73–78, 2003. URL: <http://www.isi.edu/info-agents/workshops/ijcai03/papers/Cohen-p.pdf>.
- 5 G. Correndo, M. Salvadores, I. Millard, H. Glaser, and N. Shadbolt. SPARQL query rewriting for implementing data integration over linked data. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, NY, USA, 2010. ACM. doi:10.1145/1754239.1754244.
- 6 Mathieu d'Aquin and Enrico Motta. Extracting Relevant Questions to an RDF Dataset Using Formal Concept Analysis. In *Proc. of the 6th K-CAP*, USA, 2011. doi:10.1145/1999676.1999698.
- 7 Géraud Fokou, Stéphane Jean, Allel HadjAli, and Mickaël Baron. RDF query relaxation strategies based on failure causes. In *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Proceedings*, pages 439–454, 2016. doi:10.1007/978-3-319-34129-3_27.
- 8 Andrey Gubichev and Thomas Neumann. Exploiting the query structure for efficient join ordering in SPARQL queries. In *17th International Conference on Extending Database Technology, EDBT 2014*, pages 439–450, 2014. doi:10.5441/002/edbt.2014.40.
- 9 F. Picalausa and S. Vansummeren. What Are Real SPARQL Queries Like? In *Proceedings of, SWIM '11*, pages 7:1–7:6, New York, NY, USA, 2011. ACM. doi:10.1145/1999299.1999306.
- 10 B. R. Kuldeep Reddy and P. Sreenivasa Kumar. Efficient approximate SPARQL querying of Web of Linked Data. In *6th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2010), collocated with ISWC-2010*, pages 37–48, 2010. URL: <http://ceur-ws.org/Vol-654/paper4.pdf>.
- 11 Umberto Straccia and Raphaël Troncy. Towards Distributed Information Retrieval in the Semantic Web: Query Reformulation Using the oMAP Framework. In *3rd European Semantic Web Conference, ESWC 2006, Proceedings*, pages 378–392, 2006. doi:10.1007/11762256_29.
- 12 Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadís Martínez, Javier Sierra, and Axel Polleres. Efficiently Joining Group Patterns in SPARQL Queries. In *7th Extended Semantic Web Conference, ESWC 2010, Proceedings, Part I*, pages 228–242, 2010. doi:10.1007/978-3-642-13486-9_16.
- 13 Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. From keywords to semantic queries - Incremental query construction on the Semantic Web. *J. Web Sem.*, 7(3):166–176, 2009. doi:10.1016/j.websem.2009.07.005.