# Query Generation for Semantic Datasets

Jeff Z. Pan and Yuan Ren
Department of Computing Science
University of Aberdeen

Honghan Wu
College of Computer and Software
Nanjing University of Information and Technology

Man Zhu
School of Computer Science & Engineering
Southeast University, China

## ABSTRACT

Due to the increasing volume of and interconnections between semantic datasets, it becomes a challenging task for novice users to know what are included in a dataset, how they can make use of them, and particularly, what queries should be asked. In this paper we analyse several types of candidate insightful queries and propose a framework to generate such queries and identify their relations. To verify our approach, we implemented our framework and evaluated its performance with benchmark and real world datasets.

## 1. INTRODUCTION

Recent years have experienced a rapid growth of RDF data published as Linked Data,[1] where OWL ontologies are used to annotate and connect data. Semantic data can be exploited by queries [12], such as those in the standard RDF query language SPARQL.

It becomes a challenging task for novice users to know what are included in a dataset and how they can be used. Firstly, novice users tend not to be familiar with RDF and SPARQL. Secondly, users are likely to be unfamiliar with external datasets that are linked to their local ones. Given some target dataset(s), it would be desirable to have service to recommend insightful queries to users. For example, in the Lehigh University Benchmark (LUBM),[2] the following queries Q1 and Q2 have the same results under RDF simple interpretation (i.e. without reasoning [7, 4]).

```
#Q1: Return those who take a Course
SELECT ?x WHERE {
  ?x lubm:takeCourse ?y .
  ?y rdf:type lubm:Course .  }
#Q2: Return Undergraduates who take a Course
SELECT ?y WHERE {
  ?x lubm:takeCourse ?y .
  ?x rdf:type lubm:Undergraduate .
  ?y rdf:type lubm:Course .  }
```

This implies that only undergraduate students are taking courses in the dataset, which might be somehow *surprising*, as post-graduate students are *not* taking any courses. This gives users a sense of the quality of the query answers when reasoning is disabled.

[1]http://wifo5-03.informatik.uni-mannheim.de/lodcloud/state/
[2]http://swat.cse.lehigh.edu/projects/lubm/

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*K-CAP* '13, June 23-26, 2013, Banff, Canada.
Copyright 2013 ACM 978-1-45-03-2102-0/13/06 ...$15.00.

Query generation (QG) has been studied in the field of database, based on database schemas (e.g. [13]) or actual data (e.g. [9]), where the main motivation is to generate queries for testing databases. A related research problem is query recommendation (QR), where query logs are widely used to generate queries based on querying and browsing behaviours of users [2, 15]. Similar to QG for databases, there exists work on QG for testing semantic web engines [6, 5], based on ontologies or parameterisations. d'Aquin and Motta [3] proposed a QG approach based on formal concept analysis, which uses computationally complex ontological reasoning. In this paper, we propose a tractable query generation approach based on data summarisation and graph patterns.

## 2. GRAPH AND GRAPH PATTERN

We assume the readers have basic knowledge about RDF and SPARQL. An RDF graph $G$ can be divided into mutually disjoint schema graph $G_s$ and instance graph $G_i$. An RDF graph is an instance graph if it only contains *type triples* of form <$x$, $rdf{:}type$, $A$> or relation triples <$y$, $R$, $z$> , where $A$ is a class, $R$ is a user defined property and $x, y, z$ are resources. With these notions, to facilitate query generation, we define a graph as follows:

*Definition 1.* **(Graph)** A *labeled, directed multiple graph* (graph for short) $G = \langle N, E, M, L\rangle$ is a four-tuple, where $N$ is the set of nodes, $E$ is the set of edges, $M : E \to N \times N$ maps an edge to an ordered pair of nodes, $L$ is the labelling function that for each node $n \in N$, its label $L(n)$ is a set of URI references, and for each edge $e \in E$, its label $L(e)$ is a URI reference.

For any RDF instance graph $\mathcal{D}$, let $T$ be its set of types and $TT$ be its set of type triples, a unique graph $G_{\mathcal{D}} = \langle N, E, M, L\rangle$ can be constructed as follows, where type triples are aggregated as the labels of nodes, and relation triples are represented as directed, labelled edges in the graph: (i) $N = \{x| < x, p, o >\in \mathcal{D}$ or $< s, p, x >\in \mathcal{D}\} \setminus T$; (ii) $E = \mathcal{D} \setminus TT$; (iii)$\forall < s, p, o >\in E, M(< s, p, o >) = (s, o), L(< s, p, o >) = p$; (iv) $\forall n \in N, L(n) = \{C| < n, rdf{:}type, C >\in \mathcal{D}\}$.

*Definition 2.* **(Graph Operations)** A graph $\langle N_1, E_1, M_1, L_1\rangle$ is a *sub-graph* of another graph $\langle N_2, E_2, M_2, L_2\rangle$ IFF $N_1 \subseteq N_2$, $E_1 \subseteq E_2$, $\forall e \in E_1$, $M_1(e) = M_2(e)$, $L_1(e) = L_2(e)$ and $\forall n \in N_1$, $L_1(n) \subseteq L_2(n)$.

A graph $G$ is the *intersection* of graphs $G_1$ and $G_2$ IFF it is the largest sub-graph of both $G_1$ and $G_2$.

Two graphs $\langle N_1, E_1, M_1, L_1\rangle$ and $\langle N_2, E_2, M_2, L_2\rangle$ have a *union* IFF $\forall e \in E_1 \cap E_2$, $M_1(e) = M_2(e)$, $L_1(e) = L_2(e)$. Their *union* is a graph $\langle N, E, M, L\rangle$ such that $N = N_1 \cup N_2$, $E = E_1 \cup E_2$,

$\forall e \in E_1 \cup E_2$, $M(e) = M_1(e)$ or $M(e) = M_2(e)$, $L(e) = L_1(e)$ or $L(e) = L_2(e)$, and $\forall n \in N_1 \cup N_2$, $L(n) = L_1(n) \cup L_2(n)$.

SPARQL supports many different patterns. In this paper, we are interested in basic graph pattern (BGP) and the `FILTER NOT EXISTS`. A BGP is a set of triples with variables. A solution to a BGP is a mapping of the variables to resources or blank nodes in the RDF graph, s.t. the mapped graph is a subset of the RDF graph. For a BGP `bgp`, the solution to `FILTER NOT EXISTS bgp` are the mappings of variables to resources or blank nodes, s.t. the mapped triples do not occur in the RDF graph. We also slightly abuse the notion to say that, two queries $Q_1$ and $Q_2$ can be combined into a composite query, denoted by $Q_1 \wedge Q_2$, by including the variables and triples in both $Q_1$ and $Q_2$.

In this paper we are interested in *counjunctive queries without non-distinguished variables* and their complements and composites. A conjunctive query BGP contains only the type triples and the relation triples, where only the subject of type triples, and subject and object of relation triples can be variables. Queries with non-distinguished variables are supported by the new SPARQL standard, where all variables in queries must be bounded to named entities in the RDF graph. With the above considerations, BGP queries discussed in this paper can be regarded as a special kind of graph:

*Definition 3.* **(Graph Pattern)** A *graph pattern* is a graph in which some nodes are variables.

For any conjunctive query BGP, a unique graph pattern can be constructed, and vice versa. We use $Q(GP)$ to denote a query constructed from a graph pattern $GP$. It is obvious that $Q(GP_1) \wedge Q(GP_2) = Q(GP)$, where $GP$ is the union of $GP_1$ and $GP_2$.

Query answering can also be realised by graph pattern matching:

*Definition 4.* **(Instance)** A *substitution* $\S = (v_1 \rightarrow v_2)$ replaces a vector of variables $v_1$ with a vector of URI references/literals/blank nodes/variables $v_2$. A substitution is variable-free IFF $v_2$ contains no variable. A graph $G$ is an *instance* of a graph pattern $G'$, denoted by $G : G'$, IFF there exists a substitution $\S$ such that $G'_{\S} = G$. Given a dataset $\mathcal{D}$, we use $I_{\mathcal{D}}(G')$ to denote the set of all subgraphs of $\mathcal{D}$ that are instances of $G'$. Obviously, $G \in I_{\mathcal{D}}(G)$ since an empty substitution $(\emptyset \rightarrow \emptyset)$ exists. And $I_{\mathcal{D}}(G) = \{G\}$ when $G$ contains no variable. A graph $G$ is an *$v$-instance* of a graph $G'$ w.r.t. $\mathcal{D}$ and a vector of variables $v$ IFF there is a variable-free substitution $(v \rightarrow v')$ such that $G'_{(v \rightarrow v')} = G$, and $I_{\mathcal{D}}(G) \subseteq I_{\mathcal{D}}(G')$. We use $I_{\mathcal{D},v}(G)$ to denote the smallest set of all $v$-instances of $G$ w.r.t. $\mathcal{D}$ and $v$.

When the $\mathcal{D}$ is clear from context, we omit it in the notations.

The following proposition shows the relation between a solution of a query and an instance of the graph pattern of the query:

**Proposition 1** *(Query Answering) Given a dataset $\mathcal{D}$, for any counjunctive query BGP Q of a vector of variables $v_1$, let GP be its corresponding graph pattern, then if the mapping from $v_1$ to $v_2$, a vector of resources, is a solution of Q w.r.t. $\mathcal{D}$, then $GP_{(v_1 \rightarrow v_2)} \in I(GP)$. And if $G \in I(GP)$ with substituition $(v_1 \rightarrow v_2)$, then the mapping from $v_1$ to $v_2$ is a solution of Q w.r.t. $\mathcal{D}$.*

With the above notations, we can investigate query generation problem by investigating graph patterns in datasets.

## 3. CANDIDATE INSIGHTFUL QUERIES

Given a target graph $\mathcal{D}$, the generation of *candidate insightful queries* for $\mathcal{D}$ can be regarded as a process of identifying typical graph patterns (or typical graph pattern pairs) having instances within $\mathcal{D}$, such that these typical graph patterns (or typical graph pattern pairs) provide users some insights about the structure of $\mathcal{D}$.

### 3.1 Typical Graph Patterns

Typical graph patterns are concerned with the structured relations among domain objects. While schema graphs (or ontologies) specify some global structure, typical instance graph patterns inform users some possibly additional structure in the current version of the graph. There can be different kinds of typical graph patterns, such as star-shaped graphs, shallow tree shaped graphs, deep tree shaped graphs and graphs with loops.

*Definition 5.* **(Looped Graph Pattern)** A graph pattern is a *looped graph pattern* if it contains a circle of nodes.

A looped graph pattern is a *variable-looped graph pattern* if it contains a circle of variables.

A query is a *(variable-)looped query* if its corresponding graph pattern is a (variable-)looped graph pattern.

In this paper, we are particularly interested in graphs with looped graph pattern, since loops reveal the multiplicity of the connections between objects, i.e. objects are connected in the dataset via multiple paths. While nominal-free[3] ontologies are not sufficiently expressive to accurately represent loops. In other words, graphs with loops might give users some insights on complex relations among objects that are not captured in the corresponding ontology.

In what follows, we first introduce the notion of graph pattern correspondence and then will revisit looped queries.

### 3.2 Graph Pattern Correspondence

Graph pattern correspondence is concerned with the relationships between two group of objects. In the Introduction, we gave some example of queries with the same set of answers. They can be formally described by the correspondence of two graph patterns as defined below:

*Definition 6.* **(Graph Pattern Correspondence)** Given an RDF instance graph $\mathcal{D}$, two graph patterns $GP_1$ and $GP_2$ correspond on a vector of variables $v$ IFF there is a variable-free substitution $v \rightarrow v'$ such that $GP_{1(v \rightarrow v')} \in I_{v,\mathcal{D}}(GP_1)$ and $GP_{2(v \rightarrow v')} \in I_{v,\mathcal{D}}(GP_2)$. $v'$ is called the $v$-correspondence of $GP_1$ and $GP_2$ w.r.t. $\mathcal{D}$.

We use $C_{\mathcal{D},v}(GP_1, GP_2) = \{v' | v'$ is a $v$-correspondence of $GP_1$ and $GP_2$ w.r.t. $\mathcal{D}$ $\}$ to denote the set of all $v$-correspondences. In the rest of the paper, we omit $\mathcal{D}$ from the notations when it is clear from context.

From the above definition, it is obvious that two graph patterns $GP_1$ and $GP_2$ correspond on a vector of variables $v$ IFF there is a solution to $Q(GP_1)$ and a solution to $Q(GP_2)$ that have the same value assigned to $v$. While $C_{\mathcal{D},v}(GP_1, GP_2)$ actually indicates the different values of $v$ that can be shared by solutions of $Q(GP_1)$ and $Q(GP_2)$. The following theorem shows the relation between graph pattern correspondence and conjunctive query answering:

**Theorem 1** *Two graph patterns $GP_1$ and $GP_2$ corresponds on variables v IFF $Q(GP_1) \wedge Q(GP_2)$ has a solution.*

*$C_{\mathcal{D},v}(GP_1, GP_2) = \{v' | v'$ is the value assigned to $v$ in some solution of $Q(GP_1) \wedge Q(GP_2)$ w.r.t. $\mathcal{D}\}$.*

This result can be utilised to generate the following kinds of insightful queries.

[3]Nominal is one of the OWL features that could introduce scalability problem for reasoning.

**1. Queries with strong Correspondence:** For two graph patterns $GP_1$ and $GP_2$, let $v$ be a vector of all their shared variables, $Q(GP_1)$ and $Q(GP_2)$ are insightful queries with strong correspondence if $|C_v(GP_1, GP_2)|$ is higher or lower enough w.r.t. $|I_v(GP_1)|$ or $|I_v(GP_2)|$, which indicates that $Q(GP_1)$ and $Q(GP_2)$ share a lot, or very few solutions on variables in $v$, respectively.

This is because, $|I_v(GP_1)|$ and $|I_v(GP_2)|$ are the number of different solutions assigned to $v$ in $Q(GP_1)$ and $Q(GP_2)$, respectively, and $|C_v(GP_1, GP_2)|$ is the number of different solutions assigned to $v$ in both $Q(GP_1)$ and $Q(GP_2)$. When $\frac{|C_v(GP_1,GP_2)|}{I_v(GP_1)}$ is close to 1, it indicates that most of the solutions to $Q(GP_1)$ can also be regarded as solutions to $Q(GP_2)$ for all variables they share. When it is close to 0, it indicates that only very few solutions to $Q(GP_1)$ can be regarded as solutions to $Q(GP_2)$. Both queries with high shared solutions and the queries with low shared solutions as insightful queries.

Note that such a solution-sharing relation is not symmetric, i.e. it is possible that $Q(GP_1)$ shares many solutions with $Q(GP_2)$ but $Q(GP_2)$ only shares a few with $Q(GP_1)$.

**2. Queries on Exceptions:** With the correspondence defined in Definition 6 we can generate insightful queries due to exceptions.

- For a pair of queries $Q_1$ and $Q_2$ with very high correspondence, $Q_1 \wedge \{$`FILTER NOT EXISTS` $(Q_2)\}$ (or $Q_2 \wedge \{$`FILTER NOT EXISTS` $(Q_1)\}$) will be an insightful query on exceptions. Obviously, if two queries share a lot solution, then the solutions that do not belong to both of them are quite interesting to users.
- For a pair of queries $Q_1$ and $Q_2$ with strong low correspondence, $Q_1 \wedge Q_2$ will also be an insightful query on exceptions. If two queries share very few solutions, then the solutions that belong to both of them are interesting to users.

An extreme case of queries on exception is empty queries. In this paper, an empty query is a query that does not have any solution on the given RDF dataset, rather than arbitrary datasets [1], in which a query is empty if it does not have any solution for any dataset. Our notion of emptiness is based on the input instance graph(s), while their notion of emptiness is forced by the input ontology (schema graph). Our notion of empty queries is weaker and cannot be checked by their approach.

Now that we introduce the notion of graph pattern correspondence. Let us revisit looped queries, some of which can be regarded as a special extension of queries with high correspondence: for two graph pattern $GP_1$ and $GP_2$, if there is a path of variables $v_1, v_2, \ldots, v_n$ in $GP_1$ and a path of variables $u_1, u_2, \ldots, u_m$ in $GP_2$, $v_1 = u_1$ and $v_n = u_m$ are variables shared by $GP_1$ and $GP_2$, and $GP_1$ and $GP_2$ have high correspondence w.r.t. vector $\langle v_1, v_n \rangle$, then $Q(GP_1) \wedge Q(GP_2)$ is a looped query.

This is obvious, since by combining $GP_1$ and $GP_2$ we have a looped graph pattern containing variable loop $v_1, v_2, \ldots, v_n, u_{m-1}, \ldots, u_2, v_1$. And this looped graph pattern can be transformed to $Q(GP_1) \wedge Q(GP_2)$. If $GP_1$ and $GP_2$ have high correspondence, it indicates that the solutions to $Q(GP_1)$ construct loop structures with solutions to $Q(GP_2)$. Such loop structures are captured by solutions of the looped query $Q(GP_1) \wedge Q(GP_2)$.

## 4. QUERY GENERATION FRAMEWORK

Our framework is depicted in Figure. 1. The first step identifies the graph patterns in datasets and extract their corresponding instances. The input of this step include the datasets and optionally some related constraints, such as size on the graph patterns. The output of this step is a set of pairs $\langle GP, I_{\mathcal{D}}(GP) \rangle$, where $GP$ is a graph pattern and $\mathcal{D}$ is a dataset. The main challenge is that there can be *too many graph patterns with useless 0 correspondence*. To avoid generating such meaningless query pairs, we make sure that the shared variables of two queries (or graph patterns) belong to the same type. Hence, we perform data summarisation based on the types of nodes in the graph. Given an RDF graph, the summarisation is to generate a condensed description which reduces the search space of the graph pattern mining. Roughly speaking, the summarisation is an analogue to the schema, e.g. E-R diagram, in relational database system. This summarisation takes the form of graph patterns which reveal the possible relations among individuals. One of its good properties is that any query which is interesting by our definitions is a subgraph( or subgraphs) of the summarisation. This property allows us jump out from the "swamp" of original data graphs and focus on the summarisations to mine interesting queries. Furthermore, the size of the summarisation graph is extremely small by comparing to the original graph. The biggest summarisation graph in our test datasets only has 44 triples. Such tiny sized summarisations can largely facilitate the mining process e.g. expensive mining algorithms are applicable.

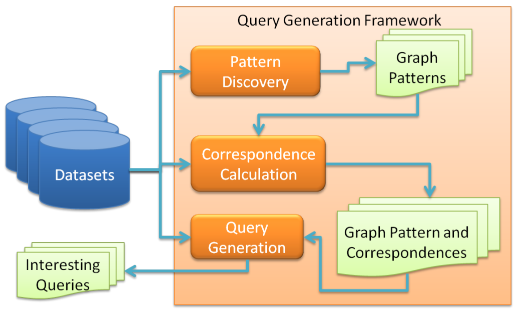
Query Generation Framework
Pattern Discovery
Graph Patterns
Datasets
Correspondence Calculation
Query Generation
Graph Pattern and Correspondences
Interesting Queries

**Figure 1: Query Generation Framework.**

The second step computes the correspondences between graph patterns. The input of this step is the data summaries delivered by the previous step and the datasets. The output of this step is a set of 4-tuples $\langle GP_1, GP_2, support, confidence \rangle$ where $GP_1$ and $GP_2$ are graph patterns, $support$ and $confidence$ are used to characterise the correspondences between $GP_1$ and $GP_2$. Assuming $GP_1$ and $GP_2$ share a vector of variables $v$, then $support = |C_v(GP_1, GP_2)|$, $confidence = \frac{|C_v(GP_1,GP_2)|}{|I_v(GP_1)|}$ when $I_v(GP_1) > 0$ and 1 when $I_v(GP_1) = 0$. $support$ indicates how frequent do $GP_1$ and $GP_2$ share instances w.r.t. $v$. The higher $support$ is, the more frequent the two graph patterns share instances on $v$ in general. $confidence$ indicates how frequent do instances of $GP_1$ w.r.t. $v$ are also instances of $GP_2$. The higher $confidence(GP_1, GP_2)$ is, the more frequent that instances of $GP_1$ can be shared with $GP_2$. The challenge in this step is to *identify different patterns with desired correspondence*. In the domain of ontology learning, algorithms of inductive logic programming (ILP), association rule mining have been explored to find relationships between concepts and relations [14, 8]. Inspired by these works, we examine three different approaches (worst case polynomial time) to find the corresponding graph patterns. **FOIL** (First Order Inductive Learning) constructs the graph pattern by including a set of possible reachable variables via graduately growing the graph pattern. The algorithm selects a best variable from the set by a gain function (c.f. [11]). FOIL tends to generate star-shaped graph patterns. **COMB** and **LOOP** approaches utilise the association rule mining technology. They tend to generate chain-shaped or looped graph patterns.

This third step generates insightful queries based on our discussion in Sec. 3. The input of this step is the datasets, and the computed correspondences between graph patterns. The output of this step will be a set of insightful queries or query pairs.

## 5. EVALUATION

We implemented the framework in Sec. 4 and evaluate its performance with benchmark datasets: **LUBM** (Lehigh University Benchmark) is an artificial dataset in which data is automatically generated. Its transparency makes it easier for us to examine whether the queries genreated by our system is useful or not. We generated 15,247 triples in our evaluation. **DBLP** is a large and real world dataset which includes bibliography data. In our evaluation, we used DBLP2011 data [4] (3,584,734 triples). **DBTune** hosts a selection of music-related RDF datasets. In our evaluation, we used the Jamedon [5] (1,047,950 triples) and BBC-PEEL [6] (271,369 triples) datasets. They are both using the music ontology [7] and the FOAF ontology [8] as terminologies.

We applied our framework on the above datasets and generate queries on graph patterns with strongest support and confidence. Such generation can be performed efficiently. We examined the top 20 generated queries (query pairs) for each dataset and results showed that they are all meaningful. Some examples of generated queries from the LUBM dataset are presented in Table 1.

**Table 1: Query examples.**

| | Query correspondences |
|---|---|
| 1 | `SELECT ?x` |
| | `WHERE {?x rdf:type lubm:ResearchGroup.}` |
| | |
| | `SELECT ?x` |
| | `WHERE {?x lubm:subOrganizationOf ?y.` |
| | `        ?y rdf:type lubm:Department.}` |
| | Query exceptions |
| 2 | `SELECT ?x` |
| | `WHERE {?x lubm:headOf ?y.` |
| | `        ?y rdf:type lubm:Department.` |
| | `FILTER NOT EXISTS` |
| | `       {?x rdf:type lubm:FullProfessor}}` |
| | Looped queries |
| 3 | `SELECT ?x ?y ?z ?o` |
| | `WHERE {?x lubm:worksFor ?z.` |
| | `        ?x lubm:teacherOf ?y.` |
| | `        ?o lubm:memberOf ?z.` |
| | `        ?o lubm:takesCourse ?y.` |
| | `        ?o lubm:advisor ?x.}` |

For example, query 1 suggests a high correspondence between the sub-organizations of some department and research groups. This reveals that a sub-organization of a department is very likely to be a research group. Such insights will be helpful when investigating the administration structures in universities. Query 2 investigates if the head of a department must be a full professors. In the evaluated dataset this indeed is the case. By automatically generating a query on cases where head of department is not known to be a full professor, users can easily identify potential exceptions. Another category of insightful queries are queries with loops, such as Query 3 in Table 1. This query suggests very high support and confidence that an advisor and an advisee work for (is a member of) the same entity, and the advisor is the teacher of some course that is taken by the advisee. This query contains three loops, and is very difficult to be expressed with normal ontology language.

[4]`http://law.di.unimi.it/webdata/dblp-2011/`
[5]`http://dbtune.org/jamendo/`
[6]`http://dbtune.org/bbc/peel/`
[7]`http://musicontology.com/`
[8]`http://www.foaf-project.org/`

## 6. CONCLUSION AND FUTURE WORK

We presented a novel and tractable approach to generating candidate insightful queries for semantic datasets. Combination of data summarisation and different mining technologies have been exploited to extract graph patterns and construct candidate insightful queries. Our evaluation shows that the proposed framework can generate insightful queries from synthetic and real world datasets.

Apart from QG based on the systematic parametrisation [5], we are currently exploring how to extend our framework to embed the support of tractable reasoning and schema [10] into our framework, so that, on the one hand, reasoning can be done on the fly with data summarisation; on the other hand, reasoning can be used to eliminate query pairs that are inferred to share all (or no) solutions.

**Acknowledgement** The work is supported by the EC Marie Curie K-Drive project (286348).

## 7. REFERENCES

[1] F. Baader, M. Bienvenu, C. Lutz, F. Wolter, et al. Query and predicate emptiness in description logics. *Proc. of KR2010.*

[2] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *Proc. of SSDBM 2009*, pages 3–18, 2009.

[3] M. d'Aquin and E. Motta. Extracting Relevant Questions to an RDF Dataset Using Formal Concept Analysis. In *Proceedings of the sixth international conference on Knowledge capture*, pages 121–128, 2011.

[4] A. Fokoue, F. Meneguzzi, M. Sensoy, and J. Z. Pan. Querying linked ontological data through distributed summarization. In *AAAI2012*.

[5] O. Görlitz, M. Thimm, and S. Staab. Splodge: Systematic generation of sparql benchmark queries for linked open data. In *Proc. of ISWC 2012*, pages 116–132. Springer, 2012.

[6] B. C. Grau and G. Stoilos. What to Ask to an Incomplete Semantic Web Reasoner? In *IJCAI*, pages 2226–2231, 2011.

[7] N. Heino and J. Z. Pan. RDFS Reasoning on Massively Parallel Hardware. In *Proc. of ISWC2012*.

[8] J. Lehmann, S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9:71–81, 2011.

[9] C. Mishra, N. Koudas, and C. Zuzarte. Generating Targeted Queries for Database Testing. In *Proc. of SIGMOD*, 2008.

[10] J. Z. Pan, E. Thomas, Y. Ren, and S. Taylor. Tractable Fuzzy and Crisp Reasoning in Ontology Applications. In *IEEE Computational Intelligence Magazine*, 2012.

[11] J. Quinlan and R. Cameron-Jones. Foil: A midterm report. In *Machine Learning: ECML-93*, pages 1–20. Springer, 1993.

[12] W. Siberski, J. Z. Pan, and U. Thaden. Querying the semantic web with preferences. In *Proc. of ISWC2006*.

[13] D. Slutz. Massive Stochastic Testing of SQL. In *VLDB*, pages 618–622, 1998.

[14] J. Völker and M. Niepert. Statistical schema induction. *The Semantic Web: Research and Applications*, 2011.

[15] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *Proc. of WWW2006*.