SPLODGE: Systematic Generation of SPARQL Benchmark Queries for Linked Open Data

Olaf Görlitz, Matthias Thimm, and Steffen Staab

Institute for Web Science and Technology
University of Koblenz-Landau, Germany
{goerlitz,thimm,staab}@uni-koblenz.de
http://west.uni-koblenz.de/

Abstract. The distributed and heterogeneous nature of Linked Open Data requires flexible and federated techniques for query evaluation. In order to evaluate current federation querying approaches a general methodology for conducting benchmarks is mandatory. In this paper, we present a classification methodology for federated SPARQL queries. This methodology can be used by developers of federated querying approaches to compose a set of test benchmarks that cover diverse characteristics of different queries and allows for comparability. We further develop a heuristic called SPLODGE for automatic generation of benchmark queries that is based on this methodology and takes into account the number of sources to be queried and several complexity parameters. We evaluate the adequacy of our methodology and the query generation strategy by applying them on the 2011 billion triple challenge data set.

1 Introduction

The Linked Data cloud offers a huge amount of machine readable, structured data and its full potential can only be leveraged by querying over multiple data sources. Therefore, efficient query processing on the Linked Data cloud is currently an active research area and different novel optimization approaches have been published [1,9,15,29,31]. As there is currently no common benchmark for federated Linked Data query evaluation, different datasets and different queries are being employed for the evaluation, which often prevents a direct comparison of approaches. A common benchmark based on actual queries on the Linked Data cloud could solve this problem. But since applications for Linked Data query processing are not in wide use yet such query collections are currently not available. Hence, the use of real queries in a Linked Data benchmark is not a viable option anytime soon.

Benchmarks serve different purposes. A major objective is to compare the performance of different implementations. Moreover, the quality of a system can be assessed by testing common cases and corner cases, e. g. queries with high complexity or queries which generate large intermediate result sets. Artificial datasets are usually highly structured [6] and allow for a well controlled evaluation environment. But an adaptation mimicking the Linked Data characteristics

P. Cudré-Mauroux et al. (Eds.): ISWC 2012, Part I, LNCS 7649, pp. 116–132, 2012.

is not straightforward. Besides, benchmarks like SP²Bench [27], LUBM [10], or BSBM [4] are typically designed for evaluating centralized RDF stores and we consider them inappropriate for the evaluation of query processing across Linked Data. Evaluation approaches based on real data can use hand-crafted queries, like in FedBench [26], or automatically generated queries as in [11]. Either way, the queries should expose characteristics which are assumed to cover a sufficiently large variety of real queries. However, meaningful queries can only be generated manually with a lot of effort because the content of the data sources needs to be analyzed in advance. In contrast, automatic query generation is less tedious and can produce many queries with specific characteristics, even for varying data sources as in the Linked Data cloud.

In this paper we abstract from specific query interfaces, i. e. query processing could be based on SPARQL endpoints, URI resolution, or the integration of data dumps. Our contribution is a methodology and a toolset for the systematic generation of SPARQL queries which cover a wide range of possible requests on the Linked Data cloud. A classification of query characteristics provides the basis for the query generation strategy. The query generation heuristic of SPLODGE (SPARQL Linked Open Data Query Generator) employs stepwise combination of query patterns which are selected based on predefined query characteristics, e. g. query structure, result size, and affected data sources. Constant values are randomly chosen and a verification step checks that all constraints are met.

In the following, we start with a review of related work. Then, in Section 3, we provide some necessary background information on both RDF and SPARQL. In Section 4 we conduct a thorough investigation of query characteristics in the context of Linked Data and, in Section 5, we continue with the presentation of our query generation approach SPLODGE. We give some insights on the implementation of our system in Section 6 and evaluate our approach in Section 7. In Section 8 we conclude with a summary and some final remarks.

2 Related Work

Federated SPARQL query processing is receiving more attention lately and a number of specific approaches have already been published, e. g. [30,23,11,25,12,15,13,29,9]. Stuckenschmidt et al. [30] employ indices for matching path patterns in queries while Harth et al. [11] use a (compressed) index of subjects, predicates, and objects in order to match queries to sources. DARQ [23] and SPLENDID [9] make use of statistical information (using hand-crafted data source descriptions or VOID [2], respectively) rather than (indices of) the content itself. FedX [29] focuses on efficient query execution techniques using chunked semi-joins. Without any precomputed statistics and a source selection based on SPARQL `ASK` queries it solely relies on join order heuristics for the query optimization. Recent work by Buil-Arada et al. [5] investigates the complexity and optimization of SPARQL 1.1 federation queries where data sources are already assigned to query expressions.

The evaluation of the above approaches is usually conducted with artificial datasets or real datasets using hand-crafted queries. LUBM [10] was one of the

first benchmarks for evaluating (centralized) RDF triple store implementations. It allows for generating synthetic datasets of different sizes representing relations between entities from the university domain. The Berlin SPARQL Benchmark (BSBM) [4] and the SP^{2}Bench [27] are more recent benchmarks, as well based on scalable artificial datasets. BSBM is centered around product data and the benchmark queries mimic real user interaction. SP^{2}Bench employs the DBLP publications schema. Its data generator ensures specific characteristics of the data distribution while the benchmark queries include also less common and complex expressions like `UNION`, `FILTER`, and `OPTIONAL`. LUBM, BSBM, and SP^{2}Bench are hardly applicable for benchmarking federation systems because the data is very structured and Linked Data characteristics can not be achieved through data partitioning.

Benchmarking with real Linked Data is, for example, provided by FedBench [26]. It employs preselected datasets from the Linked Data cloud, e. g. life science and cross domain. Different query characteristics are covered with common and complex query pattern which yield in some cases many hundred thousand results. However, due to the limitation to a few hand-picked datasets and queries, FedBench lacks scalability with respect to the Linked Data cloud. DBPSB [17] employs benchmark queries which are derived from query logs of the official DBpedia endpoint. All queries are normalized, clustered, and the most frequent query patterns, including `JOIN`, `UNION`, `OPTIONAL`, solution modifiers, and filter conditions, are used as basis for a variable set of benchmark queries. It remains open if the queries, which cover only DBpedia, are representative for Linked Data. LIDAQ [31] provides benchmark queries based on crawled Linked Data. The query complexity, using either star-shaped or path-shaped join patterns, is limited to a maximum of three joins. Other query operators or additional solution modifiers are not considered. The query generator produces sets of similar queries by doing random walks of certain breadth or depth. DBPSB and LIDAQ do not consider result size or number of data sources in their query generation.

3 Background

The *Resource Description Framework* RDF is the core data representation format in the Linked Data cloud. Let U be a set of URIs, L a set of literals and B a set of blank nodes as defined in [14] with U, L and B being pairwise disjoint. The sets U, L, and B provide the vocabulary for representing knowledge according to the guidelines for publishing Linked Open Data [3]. The basic concept of knowledge representation with RDF is the *RDF triple* or *RDF statement*.

Definition 1 (RDF statement, RDF graph). *An* RDF statement *is a triple* $S \in (U \cup B) \times U \times (U \cup L \cup B)$. *An* RDF graph $\mathcal{G}$ *is a finite set of RDF statements. For an RDF statement* $S = (s, p, o)$ *the element* s *is called* subject*,* p *is called* predicate*, and* o *is called* object*.*

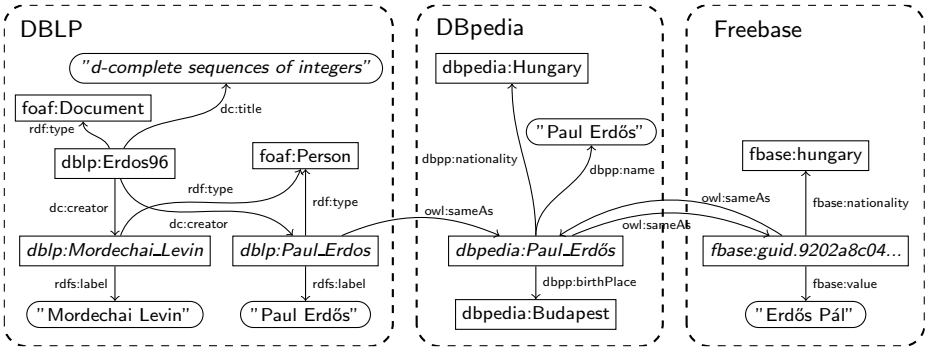
Example 1. A listing of RDF statements describing a publication by Paul Erdös (namespace definitions are omitted for better readability).

```
1 dblp:ErdosL96 rdf:type foaf:Document.
2 dblp:ErdosL96 dc:title "d−complete sequences of integers".
3 dblp:ErdosL96 dc:creator dblp:Paul_Erdos.
4 dblp:ErdosL96 dc:creator dblp:Mordechai_Levin.
5 dblp:Paul_Erdos rdf:type foaf:Person.
6 dblp:Paul_Erdos foaf:name "Paul Erdos".
7 dblp:Mordechai_Levin foaf:name "Mordechai Levin".
8 dblp:Mordechai_Levin rdf:type foaf:Person.
```

In this paper, we are interested in settings where RDF statements are distributed over a (possible large) set of different sources.

Definition 2 (Federated RDF Graph). *A* federated RDF graph $\mathcal{F}$ *is a finite set* $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$ *with RDF graphs* $\mathcal{G}_1, \ldots, \mathcal{G}_n$. *Let* $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$ *be a federated RDF graph. By abusing notation, we sometimes write* $(s, p, o) : \mathcal{G}$ *to denote that* $(s, p, o) \in \mathcal{G}$ *for* $\mathcal{G} \in \mathcal{F}$.

Example 2. We extend Example 1 (as illustrated in [8]) with RDF statements distributed across three Linked Data sources.



The *SPARQL Protocol and RDF Query Language* (or simply SPARQL) [22] is the standard query language for RDF graphs. The core notion of SPARQL are *graph patterns*. Let V be a set of variables disjoint from both U and L and $E(V)$ the set of *filter expressions* on V, cf. [22].

Definition 3 (Graph Patterns). *A* triple pattern *is a triple in* $(U \cup B \cup V) \times (U \cup V) \times (U \cup L \cup V)$ *and a* basic graph pattern *is a set of triple patterns. Every* basic graph pattern *is also a* graph pattern. *If* P_1, P_2 *are graph patterns and* $E \in E(V)$ *then* P_1 `UNION` P_2, P_1 `OPTIONAL` P_2, *and* P_1 `FILTER` E *are graph patterns as well.*

A basic graph pattern consisting of one or more triple patterns is a template that is matched in an RDF graph if all triple patterns are satisfied. Furthermore, the `UNION` combination of two patterns match if any of the two pattern matches. The `OPTIONAL` pattern matches if its first pattern matches. Additionally, further

variables might get bound if the second pattern matches as well. The FILTER pattern matches if the first pattern matches and the filter expression is satisfied.

SPARQL supports four different types of queries, namely SELECT, ASK, CONSTRUCT, and DESCRIBE queries. Their main difference is the format of the query result. Given a graph pattern P and a set $\boldsymbol{x} \subseteq V$ the query SELECT $\boldsymbol{x}$ WHERE P returns tuples of variable bindings for $\boldsymbol{x}$ such that the graph pattern P, with variables substituted accordingly, is present in the queried RDF graph. A query ASK P returns true iff the graph pattern P is satisfiable (with some variable bindings). A recent study [20] of query logs of the official DBpedia endpoint revealed that the number of CONSTRUCT and DESCRIBE queries is not significant. Therefore, we will ignore those query types in this paper. Let $BGP(P)$ be the set of basic graph patterns and $TP(P)$ be the set of triple patterns appearing in a graph pattern P. For a graph pattern P and an RDF graph $\mathcal{R}$ let $eval(P, \mathcal{G})$ be the set of possible assignments of the variables in P — i. e. functions σ of the form $\sigma : V \to U \cup B \cup L$ — such that the resulting graph pattern is satisfied in $\mathcal{G}$. We refer the interested reader to [22] for the complete semantics of SPARQL.

The SPARQL federation extension [21] introduces the two keywords SERVICE and BINDINGS and enables SPARQL queries on federated RDF graphs. While the SERVICE keyword is used for specifying RDF graphs to be queried within the federated system, the BINDINGS keyword provides means for passing the values of bounded variables to sub-queries on other RDF graphs. However, this extension only allows for the specification of federated queries when the individual RDF graphs are known to be able to answer the given sub-queries. The task of determining which RDF graphs to ask for certain sub-queries is outside the scope of the federation extension but has to be addressed by other mechanisms.

4 Parameterizing Queries

At the end of the previous section we pointed out that SPARQL querying in federated environments poses serious demands on distributed querying techniques. In order to evaluate approaches for federated SPARQL processing, multi-source queries are of major interest. Moreover, SPARQL queries used for evaluation are typically classified and parametrized along several further dimensions, e. g. with respect to complexity. However, due to the sparsity of the Linked Data cloud [24] there are today only a few real-world queries that show these characteristics. For an objective evaluation and comparison with state-of-the-art systems the developer of a federated query processing system, however, needs real data and realistic SPARQL queries. Therefore, specifically designed evaluation queries should cover common characteristics of real queries and include a sufficiently large set of SPARQL features. In the following, we develop a methodology and a toolkit that can be used by developers of approaches to federated query processing for evaluating their system in a reproducible and comparable manner. There, a developer's first step is to select and combine query parameters to fit the desired evaluation scenario, e. g. common queries and corner cases. The next step is the query generation according to the defined parameters. Finally, the evaluation

is conducted and results are presented. In order to make an evaluation reproducible, the chosen parameters and queries should be disclosed as well. In the following, we discuss different properties of SPARQL queries including aspects of distributed query processing.

We studied analysis results of real SPARQL queries [20,16,7] and features of RDF benchmarks [10,4,27,11,17] to compile query characteristics (i. e. properties) which we consider important for a query processing benchmark on Linked Data. One may argue about the choice of query characteristics. However, we do not claim completeness and like to encourage extensions of the classification parameters.

The first set of query characteristics relate to the semantic properties of SPARQL, i. e. the *Query Algebra*.

Query Type. SPARQL supports four query types, namely `SELECT`, `CONSTRUCT`, `ASK`, and `DESCRIBE`. They define query patterns in a `WHERE` clause and return a multiset of variable bindings, an RDF graph, or a boolean value, respectively. `DESCRIBE` queries can also take a single URI and return an RDF graph.

Join Type. SPARQL supports different join types, i. e. *conjunctive join* (`.`), *disjunctive join* (`UNION`), and *left-join* (`OPTIONAL`). These joins imply a different complexity concerning the query evaluation [28].

Result Modifiers. `DISTINCT`, `LIMIT`, `OFFSET`, and `ORDER_BY` alter the result set which is returned. They also increase the complexity for the query evaluation.

The next properties deal with the *Query Structure*, i. e. how basic graph patterns are combined in a complex graph pattern.

Variable Patterns. There are eight different combinations for having zero to three variables in subject, predicate, or object position of an RDF triple pattern. Some of these combinations, like bound predicate with variables in subject and/or objection position, are more common than others.

Join Patterns. Joins are defined by using the same variable in different triple patterns of a basic graph pattern. Typical join combinations are subject-subject joins (star shape) and subject-object joins (path shape). The combination of star-shaped and path-shaped joins yields a hybrid join pattern.

Cross Products. Conjunctive joins over triple patterns which do not share a common variable imply cross products. While the join parts can be evaluated independently, the cross product may involve large intermediate result sets.

The third group of properties deals with *Query Cardinality*, i. e. the number of sources, the number or joins, and the result size. Following, let P be a graph pattern, $\mathcal{F}$ be a federated RDF graph, and let $\mathcal{F}_\mathcal{P} \subseteq \mathcal{F}$ the set of *relevant data sources* for P, i. e. $\mathcal{F}_P = \{\mathcal{G} \in \mathcal{F} \mid \exists\, S \in TP(Q) : eval(S, \mathcal{G}) \neq \emptyset\}$.

Number of Sources. Our benchmark methodology is designed for query execution across Linked Data. Therefore, the number of data sources involved in query answering is an important factor, i. e. $sources(P) = |\mathcal{F}_P|$.

Number of Joins. Joining multiple triple patterns increases the complexity of a query. The number of joins $joins(P)$ is defined for basic graph patterns, i. e. conjunctive joins over a set of triple patterns, as shown below.

Query Selectivity. The proportion between the overall number of triples in the relevant graphs and the number of triples which are actually matched by query patterns is the *query selectivity* $sel(P)$. A query with high selectivity yields less results than a query with low selectivity.

$$joins(P) = \sum_{bgp \in BGP(P)} (|bgp| - 1), \qquad sel(P) = \frac{\sum_{\mathcal{G} \in \mathcal{F}_P} |eval(P, \mathcal{G})|}{\sum_{\mathcal{G} \in \mathcal{F}_P} |\mathcal{G}|}$$

According to the above characteristics, we parameterize benchmark queries such that they cover common queries and corner cases. As result of the query generation we want queries with a specific query structure which span multiple Linked Data sources. Hence, the parameters for join structure and the number of data sources involved are predominant for the query generation. However, there is a dependency between join structure and covered sources. Typical SPARQL queries have path-shaped and star-shaped join patterns or a combination thereof. Path joins span multiple data sources if two patterns are matched by different data sources but have an entity in common which occurs in subject or object position, respectively. Note that for a unique path the number of different data sources is limited by the number of joined triple pattern. Star-shaped join patterns, like {`(?x,isA,foaf:Person)`,`(?x,foaf:name,?name)`}, which match entities in multiple data sources are less interesting for Linked Data queries because they represent unions of unrelated entities.

The combination of path-shaped and star-shaped join patterns produces more complex query structures. They are supported in the query parameterization by defining join rules. These include the join combination, usually via subject or object, and the attachment position with respect to an existing path-shaped join pattern. Corner cases can exhibit a high number of joined triple patterns leading to long paths or "dense" stars. Note that, for reasons of simplicity, we do not consider joins via the predicate position and loops in the query patterns.

5 Query Generation with SPLODGE

With the definition of the query parameterization we can now go into detail of our query generation process SPLODGE. The query generator SPLODGE produces random queries with respect to the query parameters using an iterative approach. In each step, a triple pattern is chosen according to the desired query structure and added if the resulting query pattern fulfills all cardinality constraints. The iteration finishes when all structural constraints are satisfied or when they cannot be satisfied without a violation of the cardinality constraints. In the latter case, the generator cannot produce any query. As the last step, a query is modified with respect to the complexity constraints. In the following, the algorithm and the heuristics are explained in more detail.

5.1 Path Join Pattern Construction

Our query generator SPLODGE starts with the creation of path-shaped join patterns. Let $\mathcal{F} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$ be a federated RDF graph. Given a parameterization of n patterns and m sources ($m \leq n$), the algorithm constructs a sequence of triple pattern

$$PathJoin(n, m) = (t_1, \ldots, t_n) \in ((U \cup L \cup V) \times (U \cup V) \times (U \cup L \cup V))^n$$

such that

$$\forall\ i = 1, \ldots, n-1 : \quad obj(t_i) = subj(t_{i+1}) \ \text{ and } \ |\{\mathcal{G}_j \mid \exists j : t_i \in \mathcal{G}_j\}| \ = \ m.$$

If $m < n$ then several triple patterns can have the same data source. The distribution of sources among triple patterns is randomly chosen in order to allow for variations.

Computing a valid sequence $PathJoin(n, m)$ directly on the original data, is, in general, infeasible due to the huge search space in a federated RDF graph. We therefore take a heuristic approach which facilities statistical information and cardinality estimation heuristics. In order to limit the effort needed to acquire sophisticated statistics on the various data sources, we restrain our attention to path-shaped join pattern generation with bounded predicates. Note, however, that bound predicates may be replaced with variables in a post-processing step.

Definition 4 (Linked Predicate Patterns). *A linked predicate pattern l is a quadruple $l = (p_1, \mathcal{G}_1, p_2, \mathcal{G}_2)$ with $p_1, p_2 \in U$, $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{F}$, and $\mathcal{G}_1 \neq \mathcal{G}_2$. The set of valid linked predicate patterns in $\mathcal{F}$ is defined as*

$$\mathcal{L}(\mathcal{F}) = \{(p_1, \mathcal{G}_1, p_2, \mathcal{G}_2) \mid \exists\ s, o, x \in U \cup B \cup L : \ (s, p_1, x) \in \mathcal{G}_1 \ \wedge \ (x, p_2, o) \in \mathcal{G}_2\}$$

Further, we define the following sets of triples which can be matched with the first or second pattern in a linked predicate pattern

$$\phi(p_1, \mathcal{G}_1, p_2, \mathcal{G}_2) = \{(s, x) \mid \ (s, p_1, x) \in \mathcal{G}_1 \ \wedge \ (x, p_2, o) \in \mathcal{G}_2\}$$
$$\tau(p_1, \mathcal{G}_1, p_2, \mathcal{G}_2) = \{(x, o) \mid \ (s, p_1, x) \in \mathcal{G}_1 \ \wedge \ (x, p_2, o) \in \mathcal{G}_2\}$$

The combination of some $\mathcal{L}_1 = (p_1, \mathcal{G}_1, p_2, \mathcal{G}_2)$ and $\mathcal{L}_2 = (p_2, \mathcal{G}_2, p_3, \mathcal{G}_3)$ will only return results if $\tau(p_1, \mathcal{G}_1, p_2, \mathcal{G}_2) \cap \phi(p_2, \mathcal{G}_2, p_3, \mathcal{G}_3) \neq \emptyset$, i. e. bindings for the object o in the first linked predicate patterns must also be part of the join bindings x in the second linked predicate pattern, e. g. {`(?s,dc:creator,?x)`, `(?x,owl:sameAs,?o)`} and {`(?s,owl:sameAs,?y)`,`(?y,dbpp:name,?o)`} can be combined to form a path of three patterns (cf. example 2), but there will be no result if "Mordechai Levin" is bound to $?x$ because he is not included in DBpedia.

In order to compute the result size for a path-shaped join pattern we need to estimate the overlap between two arbitrary linked predicate patterns. A computation of all possible join paths is not feasible for such a large dataset.

Definition 5 (Joined Predicate Pattern Size). *For a sequence of linked predicate patterns $(p_1, \mathcal{G}_1), \ldots, (p_n, \mathcal{G}_n)$ we define the join size as*

$$\prod_{i=1..n} |\sigma_{p_i}(\mathcal{G}_i)| \ \cdot \prod_{i=2..n-1} js(p_{i-1}, \mathcal{G}_{i-1}, p_i, \mathcal{G}_i, p_{i+1}, \mathcal{G}_{i+1})$$

with $\sigma_{p_i}(\mathcal{G}_i) = \{(s, p_i, o) \in \mathcal{G}_i\}$ *and the join selectivity (js)*

$$js(p_{i-1}, \mathcal{G}_{i-1}, p_i, \mathcal{G}_i, p_{i+1}, \mathcal{G}_{i+1}) = \frac{|\tau(p_{i-1}, \mathcal{G}_{i-1}, p_i, \mathcal{G}_i)| \cdot |\phi(p_i, \mathcal{G}_i, p_{i+1}, \mathcal{G}_{i+1})|}{|\sigma_{p_i}(\mathcal{G}_i)|^2}$$

The pattern join selectivity is a value in the interval $[0, 1]$. The lower the value, the higher the selectivity of the pattern combination and the less results will be returned. In order to prevent pattern combinations that do not return any results, we prefer selectivity values closer to 1.

5.2 Star Join Pattern Construction

Star-shaped join patterns extend path-shaped join pattern at predefined *anchor points*, i. e. at a specific triple pattern in the triple pattern path. Without an anchor point the star join will represent an individual query pattern which is combined via `UNION` with the other query patterns. The query parameterization also defines the number of triple patterns in the star join and if the join variable is in subject or object position. The anchor triple pattern is automatically part of the star join. Hence, it defines the join variable, the source restriction, and the first predicate to be included in the star join.

$$StarJoin(n, \mathcal{G}) = (t_1, \ldots, t_n) \in ((U \cup L \cup V) \times (U \cup V) \times (U \cup L \cup V))^n$$

such that

$$\forall\, i = 1, \ldots, n: \quad subj(t_1) = \ldots = subj(t_n) \;\vee\; obj(t_1) = \ldots = obj(t_n) \;\text{ and}$$
$$|\{t \in StarJoin(n, \mathcal{G})\} \cap \{t \in PathJoin(k, l)\}| = 1$$

The second condition above formalizes the requirement that the star join intersects with the main path join in one triple pattern. As with path-shaped join pattern, the computation of $StarJoin(n, \mathcal{G})$ combinations on the original data is, in general, infeasible. Thus, statistics-based heuristics are also employed to combine triple patterns with bound predicates. A star-shaped join pattern will only produce results if at least one entity matches all of the triple patterns, i. e. every predicate occurs in a combination with the same entity (always in subject or object position) in the same data source. We utilize *Characteristic Sets* [18] to capture the co-occurrence of predicates with the same entities. Characteristic sets are basically equivalence classes based on distinct predicate combinations. They keep track of the number of different entities and the number for RDF triples for each predicate in the characteristic set. The latter value can be higher due to multi-value predicates. In addition, we extended the statistics with information about the data source a characteristic set occurs in.

Definition 6 (Characteristic Sets). *We define the characteristic set* $S_C(s, \mathcal{G})$ *of a subject* s *(cf. [18]) with respect to a data source* $\mathcal{G}$ *and a federated graph* $\mathcal{F}$ *with* $\mathcal{G} \in \mathcal{F}$ *via*

$$S_C(s, \mathcal{G}) := \{p \,|\, \exists\, o : (s, p, o) \in \mathcal{G}\}$$

and abbreviate $S_C(\mathcal{F}) := \{S_C(s, \mathcal{G}_i) \mid \exists\, s, p, o : (s, p, o) \in \mathcal{G}_i\}$. *Further,* reverse characteristic sets *are used to estimated the result size for star-join patterns with the join variable in object position.*

The number of results for a star-join pattern need to be taken into account for the cardinality estimation of the path-join pattern it is attached to. Therefore, we count the number of triples in source $\mathcal{G}$ which contain the subjects (or objects) of all matching characteristic sets combined with the predicate of the anchor triple pattern. The selectivity is calculated similar to Definition 5 as shown below and multiplied with the cardinality of the path-join pattern:

$$js(\mathcal{G}, (p_1, \ldots, p_n)) \;:=\; \frac{|\{(s, p_1) \mid \{p_1, \ldots, p_n\} \subseteq S_C(s, \mathcal{G})\}|}{|\sigma_{p_1}(\mathcal{G})|}$$

So far, we described how a star- join pattern is combined with a path-join pattern. This approach is extensible to combine multiple patterns and produce complex queries with mixed join patterns as depicted in Fig. 1. Due to space constraints, we do not go into further details.

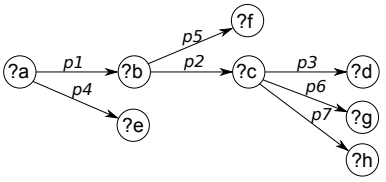


Fig. 1. Query structure generation process. First, triple patterns are combined as path-joins, i. e. (?a p_1 ?b), (?b p_2 ?c), (?c p_3 ?d), then star-joins are created for ?a, ?b, ?c.

6 Implementation

The implementation of the query generation is divided into two phases: statistics collection and the query generation based on the statistics. For our prototype implementation[1] we used the 2011 billion triple challenge dataset[2] containing about 2 billion quads, i. e. subject, predicate, object, and context. We did some pre-processing and cleanup and aggregated all contexts to their common domain name (i. e. {john,jane}.livejournal.com → livejournal.com). As a result, we reduced the 7.4 million different contexts to 789 common domains. Query patterns across different data sources are created based on these reduced domain contexts.

SPLODGE requires statistical information during query generation, i. e. for selecting triple patterns and for estimating the result size and the number of involved data sources. Due to the huge size of the Linked Data cloud, there is a trade-off between the level of statistical details and the overall space requirement for storing the meta data. Therefore, SPLODGE employs only predicate statistics, as the number of distinct predicates is much smaller compared to the number of distinct URIs in the datasets. Moreover, comprehensive statistics also impose a significant processing overhead.

[1] SPLODGE is open source and available at `http://code.google.com/p/splodge/`
[2] `http://km.aifb.kit.edu/projects/btc-2011/`

6.1 Pattern Statistics

For deciding whether triple patterns can be combined during query construction, we need information about the co-occurrence of predicates in RDF statements. For path-joins two predicates co-occur if the respective RDF statements are joined via subject/object. In addition to knowing whether predicates p_1, p_2 co-occur we also need the number $\#_{\mathcal{G}}(p)$ of RDF statements in each $\mathcal{G}$ that mention predicate p, i. e. $\#_{\mathcal{G}}(p) = |\{(s, p, o) \in \mathcal{G}\}|$.

Example 3. Following table shows co-occurrence statistics for path-joins (cf. Def. 4). Each tuple $(p_1, \mathcal{G}_1, n_1, p_2, \mathcal{G}_2, n_2)$ represents a linked predicate pattern $(s, p1, x) \in \mathcal{G}_1 \wedge (x, p2, o) \in \mathcal{G}_2$ with the respective RDF triple counts n_1 and n_2.

p_1	$\mathcal{G}_1$	n_1	p_2	$\mathcal{G}_2$	n_2
owl:sameAs	http://data.gov.uk/	22	foaf:knows	http://dbpedia.org	31
owl:sameAs	http://open.ac.uk/	58	foaf:knows	http://dbpedia.org	17
rdfs:seeAlso	http://bio2rdf.org/	15	rdf:type	http://www.uniprot.org/	38
rdfs:seeAlso	http://zitgist.com/	49	rdfs:label	http://musicbrainz.org/	36

For star-joins, we rely on *Characteristic Sets* [18]. They define equivalence classes for resources based on predicate combinations, i. e. URIs and blank nodes in subject position of RDF statements are in the same characteristic set if they have exactly the same set of predicates. Characteristic sets count the number of entities in such an equivalence class and the number of occurrences for each predicate. The latter helps to identify frequent occurrences of multi-valued predicates. We extend the characteristic sets to include the data source as well.

Example 4. The table below shows the statistical data for a specific characteristic set defined by the predicates (rdf:type, rdfs:label, rdf:sameAs). Each entry is associated with one data source $\mathcal{G}$ and contains the number of resources $\#res$ and the number of RDF triples n_i per predicate p_i in the data source.

$\mathcal{G}$	$\#res$	p_1	n_1	p_2	n_2	p_3	n_3
http://bio2rdf.org/	632	rdf:type	632	rdfs:label	844	owl:sameAs	632
http://www.uniprot.org/	924	rdf:type	924	rdfs:label	924	owl:sameAs	924
http://data.gov.uk/	1173	rdf:type	1421	rdfs:label	1173	owl:sameAs	1399

For the sake of readability, we use URIs (with namespace prefixes) in the examples above. In reality, we employ a dictionary for all predicate and source URIs and only store the entry's index number in the statistics table.

6.2 Verification

The purpose of the verification step is to ensure that all desired constraints are met by the generated queries. The query semantics and the query structure are easy to check by inspecting the syntax and query patterns of the produced SPARQL queries. In fact, these constraints are always met as the query generation is driven by the specified query structure. However, the generated queries

may not satisfy the cardinality constraints due to the estimations used by the heuristics. A reliable validation would have to execute each query on the actual data sets which is impossible because of the sheer size of the Linked Data cloud and the absence of a query processing implementation which can execute all possible types of benchmark queries in a short time. Moreover, corner case queries are intended to be hard to evaluate.

Our solution in SPLODGE is to compute a confidence value for each query. It defines how likely a query can return the desired number of results. We reject queries if their confidence value lies below a certain threshold. The confidence value is computed based on the minimum selectivity of all joins in a query.

7 Evaluation

Having explained the technical details and algorithms for the query generation we will now have a look at the evaluation of the generated queries. Evaluation in this context basically means checking if the generated queries meet the predefined cardinality constraints, i. e. if they can actually return results which were obtained from different data sources. Due to the random query generation process using cardinality estimates it is not uncommon that different queries with the same characteristics basically yield different result sizes and cover a range of various data sources. Hence, we want to evaluate two aspects: (1) how many queries in a query set fail to return any result, and (2) how good does the estimated result size match the actual results size of the queries. Furthermore, we count the number of data sources which are involved in answering a queries. To allow for an objective comparison of the effectiveness of the triple pattern selection criteria we perform the query generation with three different heuristics which are used for choosing triple patterns. For this evaluation the queries are restricted to SPARQL SELECT queries with conjunctive joins of triple patterns with bound predicate and unbound subject and unbound object.

7.1 Query Creation Heuristics

A naive approach would just randomly select predicates for use in the triple patterns of the query. For our comparison we use three different pattern selection algorithms, ranging from basic to more elaborate heuristics in order to increase the probability that the generated queries meet the desired constraints.

Baseline uses random selection of data sources and bound predicates in the data sources. It does not check if there is a connection between the data sources via the chosen predicates.

SPLODGE*lite* creates queries as described in Sec. 5. Two triple patterns are combined (in a path-join or star-join) if the statistics indicate the existence of resources which can be matched by the respective predicate combination. SPLODGE*lite* does not apply validation based on a confidence value.

SPLODGE extends SPLODGE*lite* with a computation of confidence values based on individual join selectivity. It discards queries if the confidence value is below a certain threshold, i. e. if individual joins in a query are too selective.

7.2 Setup

Representative query parameterizations are obtained by analyzing the different query structures of the FedBench queries [26]. An overview for the life-science (LS), cross-domain (CD), and linked data (LD) queries is given in Fig. 2. For simplification of the presentation, we only show the join structure and omit bound subjects/objects, filter expressions and optional parts. Queries with unbound predicate were not considered. We can basically model all of the join patterns with the parameterization described in Sec. 4. But due to space restrictions we will only consider selected queries in the evaluation. Query generation and evaluation need to be tailored for a specific dataset. We chose the 2011 billion triple challenge dataset. It contains about two billion triples and covers a large number of Linked Data sources.

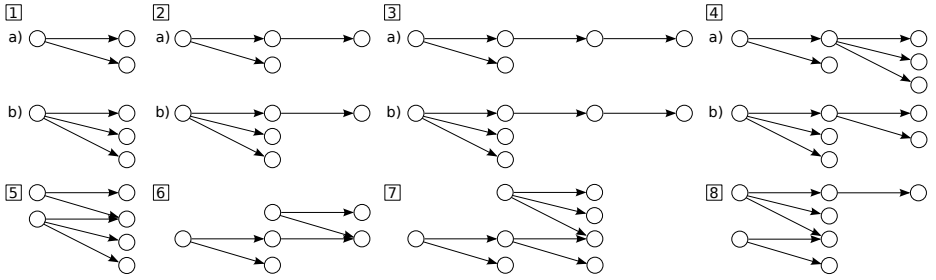


Fig. 2. Query Patterns as exposed by the FedBench queries [26] with bound predicates. Query set (1) has a single star-join, sets (2)-(4) are path-joins combined with star-joins, (5) are two combined star-joins, and sets (6)-(8) are combinations of multiple path-joins and star-joins. (1):LD[5,7], (2):LD[1,2,9,10,11], (3):CD[5,6,7],LS[3],LD[3] (4):CD[3],LD[8], (5):LS[6,7], (6):CD[4], (7):LS[4], (8):LS[5],LD[4].

The major challenge for the query generation is to produce path-join queries across different datasets. The sparsity of links between datasets makes it difficult to create long path-joins and ensure non-empty result sets. To better explore this problem space, we focused in our evaluation on path-join queries where each triple triple pattern needs to be matched by a different data source. Such queries represent interesting corner cases, as all triple patterns must be evaluated independently. Moreover, since the triple patterns have only bound predicates, many data source may be able to return results for a single triple pattern, thus increasing the number of data sources that need to be contacted. We generated sets of 100 random queries for path-joins of length 3–6 and executed them to obtain the actual number of results. All triples of the billion triple challenge dataset 2011 were loaded into a single RDF3X [19] repository.

7.3 Results

The evaluation of all queries from a query set on such a large dataset takes quite long, i. e. several hours for specific queries. The main reason is that some

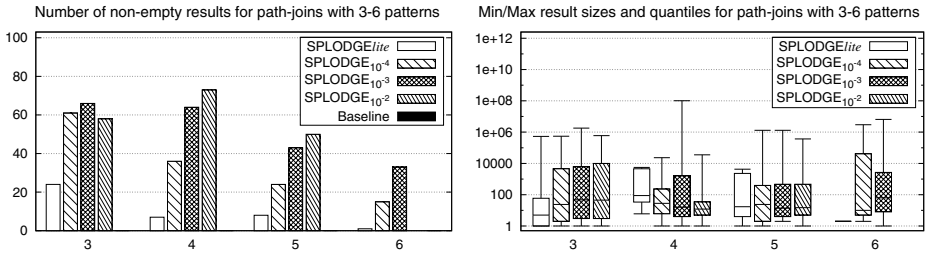


Fig. 3. Comparison of SPLENDID*lite* and SPLENDID using different confidence values, i. e. minimum join selectivity of 0.0001, 0.001, or 0.01, respectively. For each batch of 100 queries the number of non-empty results (left) and the minimum, maximum result size and the quantiles for 0.2, 0.5 and 0.8 (right) are shown.

queries produce very large (intermediate) result sets. A timeout of two minutes is set for a query to finish. For each batch of 100 queries we count the number of queries which returned non-empty results. Additionally, minimum, maximum, median, as well as 0.2 and 0.8 quantiles for all result sizes in a query batch are measured. We compare the results for SPLENDID*lite* and the regular SPLENDID query generation where the confidence value is defined by a minimum selectivity.

Figure 3 shows that SPLENDID*lite* produces only a few queries and the baseline even fails to create any query which can return results. Using the confidence value based on join selectivity increases the number of non-empty results sets significantly, i. e. from 20 to around 60 for three join patterns and by a factor of 3 to 30 for more join patterns. However, for path-joins with six triple patterns it was not possible to generate any query where the minimum join selectivity is 0.01. Considering the minimum and maximum result size we can not see any clear behavior. The minimum result size is always well below 10. The maximum goes in some case up to several million results while the 80% quantile remains below 10000 results (except for a selectivity of 0.001 and six join patterns). All query sets have a median value of less than one hundred results. The difference is smallest when the query sets have a similar number of non-empty results.

We also observe that many predicates in the queries represents schema vocabulary, e. g. `rdf:type`, `rdfs:subClassOf`, `owl:disjointWith`. This becomes even more noticeable the longer the path-join.

8 Summary and Future Work

We presented a methodology and a toolset for systematic benchmarking of federated query processing systems for Linked Data. The novel query generation approach allows for flexible parameterization of realistic benchmark queries which common scenarios and also corner cases. A thorough analysis of query characteristics was conducted to define the dimensions for the parameterization space of queries, including structural, complexity, and cardinality constraints. The implementation of SPLODGE is scalable and has proven to produce useful benchmark queries for the test dataset of the 2011 billion triple challenge.

So far, the query generation handles all predicates equally. With respect to sub-type and sub-property definitions a separate handling of schema information would allow for creating queries suitable for inference benchmarking. For future work, an extension of the statistical information would be helpful to include filter expression in the generated queries. Finally, we intend to use the benchmark queries for the evaluation of federated query processing on Linked Data.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Frame Programme under grant agreement No 257859, ROBUST.

References

1. Acosta, M., Vidal, M.-E., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 18–34. Springer, Heidelberg (2011)
2. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing Linked Datasets – On the Design and Usage of voiD, the Vocabulary Of Interlinked Datasets. In: Proceedings of the Linked Data on the Web Workshop. CEUR (2009)
3. Berners-Lee, T.: Linked Data – Design Issues. Published online (July 27, 2006), `http://www.w3.org/DesignIssues/LinkedData.html`
4. Bizer, C., Schultz, A.: The Berlin SPARQL Benchmark. International Journal on Semantic Web and Information Systems 5(2), 1–24 (2009)
5. Buil-Aranda, C., Arenas, M., Corcho, O.: Semantics and Optimization of the SPARQL 1.1 Federation Extension. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part II. LNCS, vol. 6644, pp. 1–15. Springer, Heidelberg (2011)
6. Duan, S., Kementsietsidis, A., Srinivas, K., Udrea, O.: Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets. In: Proceedings of the International Conference on Management of Data, pp. 145–156. ACM (2011)
7. Gallego, M.A., Fernández, J.D., Martínez-Prieto, M.A., de la Fuente, P.: An Empirical Study of Real-World SPARQL Queries. In: USEWOD (2011)
8. Görlitz, O., Staab, S.: Federated Data Management and Query Optimization for Linked Open Data. In: Vakali, A., Jain, L.C. (eds.) New Directions in Web Data Management 1. SCI, vol. 331, pp. 109–137. Springer, Heidelberg (2011)
9. Görlitz, O., Staab, S.: SPLENDID: Sparql Endpoint Federation Exploiting Void Descriptions. In: Proc. of the 2nd Int. Workshop on Consuming Linked Data (2011)
10. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semantics 3(2-3), 158–182 (2005)
11. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.-U., Umbrich, J.: Data Summaries for On-Demand Queries over Linked Data. In: Proceedings of the 19th International Conference on World Wide Web, pp. 411–420. ACM (2010)
12. Hartig, O., Bizer, C., Freytag, J.-C.: Executing SPARQL Queries over the Web of Linked Data. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 293–309. Springer, Heidelberg (2009)

13. Hartig, O., Langegger, A.: A Database Perspective on Consuming Linked Data on the Web. Datenbank-Spektrum 10(2), 57–66 (2010)
14. Hayes, P.: RDF Semantics. W3C Recommendation. Published online (February 10, 2004), http://www.w3.org/TR/2003/PR-rdf-mt-20031215/
15. Ladwig, G., Tran, T.: Linked Data Query Processing Strategies. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 453–469. Springer, Heidelberg (2010)
16. Möller, K., Hausenblas, M., Cyganiak, R., Grimnes, G.A., Handschuh, S.: Learning from Linked Open Data Usage: Patterns & Metrics. In: Proceedings of the Web Science Conference, pp. 1–8 (2010)
17. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.-C.: DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 454–469. Springer, Heidelberg (2011)
18. Neumann, T., Moerkotte, G.: Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins. In: 27th International Conference on Data Engineering (ICDE), pp. 984–994 (2011)
19. Neumann, T., Weikum, G.: RDF-3X: a RISC-style Engine for RDF. In: Proceedings of the 34th International Conference on Very Large Data Bases (VLDB), pp. 647–659. VLDB Endowment (2008)
20. Picalausa, F., Vansummeren, S.: What are real SPARQL queries like? In: Proceedings of the International Workshop on Semantic Web Information Management (SWIM), Athens, Greece, pp. 7:1–7:6. ACM (2011)
21. Prud'hommeaux, E., Buil-Aranda, C.: SPARQL 1.1 Federated Query. W3C Working Draft. Published online (November 10, 2011), http://www.w3.org/2009/sparql/docs/fed/service
22. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation. Published online (January 15, 2008), http://www.w3.org/TR/rdf-sparql-query/
23. Quilitz, B., Leser, U.: Querying Distributed RDF Data Sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)
24. Rodriguez, M.A.: A Graph Analysis of the Linked Data Cloud. Arxiv preprint arXiv:0903.0194, pp. 1–7 (2009)
25. Schenk, S., Staab, S.: Networked Graphs: A Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web. In: Proceedings of the 17th Int'l World Wide Web Conference, Beijing, China, pp. 585–594 (2008)
26. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 601–616. Springer, Heidelberg (2011)
27. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^2Bench: A SPARQL Performance Benchmark. In: Proceedings of the 25th International Conference on Data Engineering (ICDE), pp. 222–233 (2009)
28. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL Query Optimization. Arxiv preprint arXiv:0812.3788 (2008)

29. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 601–616. Springer, Heidelberg (2011)
30. Stuckenschmidt, H., Vdovjak, R., Houben, G.-J., Broekstra, J.: Index Structures and Algorithms for Querying Distributed RDF Repositories. In: Proceedings of the 13th Int'l World Wide Web Conference, New York, USA, pp. 631–639 (2004)
31. Umbrich, J., Hose, K., Karnstedt, M., Harth, A., Polleres, A.: Comparing data summaries for processing live queries over Linked Data. World Wide Web Journal 14(5-6), 495–544 (2011)