PFed: Recommending Plausible Federated Queries

Sara El Hassad, Hala Skaf-Molli, Pascal Molli, Florian Hacques

PFed: Recommending Plausible Federated Queries

Sara EL Hassad[1], Hala Skaf-Molli[1], Pascal Molli[1], Florian Hacques[2]

LS2N, University of Nantes, Nantes, France
(1){Sara.elhassad,Hala.Skaf,Pascal.Molli}@univ-nantes.fr,
(2)Florian.Hacques@etu.univ-nantes.fr

Abstract. Federated SPARQL query processing allows to query multiple interlinked datasets hosted by remote SPARQL endpoints. However, finding pertinent federated queries over a growing number of datasets is challenging. In this paper, we propose PFed, an approach to recommend plausible federated queries based on query logs of different datasets. The problem is not to find similar federated queries, but plausible complementary queries over different datasets. Starting with a real SPARQL query log, PFed stretches the query with real queries from different logs. A generated federated query Q is plausible if an external observer cannot deny that queries in the original logs could be subqueries of Q. PFed relies on data summaries and real SPARQL query logs to generate plausible federated queries. It starts by pruning the queries logs by selecting only queries with joinable predicates. Experimental results with real logs of DBpedia and SWDF demonstrate that PFed is able to prune drastically the logs and recommend realistic federated queries.

Keywords: Semantic Web, Consuming Linked Open Data, SPARQL query, Federated SPARQL Query, Plausible, Joinable.

1 Introduction

Following the Linked Open Data cloud (LOD) principles many datasets have been published. Federated SPARQL query engines [15,1] have been developed to query multiple interlinked datasets hosted by remote SPARQL endpoints. They allow to consume LOD data in a decentralized way without the need to copy the data. However, finding pertinent federated queries over a growing number of datasets is challenging. This requires to fully understand the datasets and find potential joins among them. In this paper, we propose PFed, an approach to recommend federated queries for end-users. More precisely, PFed recommends plausible federated queries using the query logs of different SPARQL endpoints. This is not a classical recommendation problem. In recommender systems [2], the problem is to recommend resources (or items) for users based on similar ones already seen by the users. In PFed, we start with a SPARQL query from a query log and we stretch this query with real queries from other existing query logs for generating a plausible federated query Q. The generated federated query Q is

called plausible if an external observer of PFED cannot deny that queries in the original logs could be subqueries of Q. To illustrate, Figure 1 presents extracted queries from real query logs [1] of SWDF and DBpedia. More precisely, Figure 1a presents SPARQL queries from the log of *SWDF 2012* and Figure 1b present SPARQL queries from the log of *DBpedia 3.5.1*[2].

```
Q1S: SELECT * WHERE {
       ?obj rdf:type foaf:Organization .
       ?obj foaf:based_near ?place
     }

Q3S: SELECT * WHERE {
       {?paper swrc:author ?author}
       UNION {?paper foaf:maker ?author}
       OPTIONAL {?paper swrc:abstract ?abstract}
     }
```

(a) SWDF query log

```
Q1D: SELECT * WHERE {
       ?country rdfs:label ?label .
       ?country dbpedia2:capital ?capital .
       ?capital geo:lat ?lat .
       ?capital geo:long ?long
     }

Q3D: SELECT * WHERE {
       {dbpedia:Paris ?property ?hasValue}
     UNION
       {?isValueOf ?property dbpedia:Paris}}
```

(b) DBpedia query log

Fig. 1: SPARQL queries from the logs of SWDF 2012 and DBpedia 3.5.1

Consider the $Q1S$ from the log of SWDF, this query can be extended with the query $Q1D$ from the log of DBpedia. The result is the SPARQL 1.1 federated Query $Q1S1D$ given in Figure 2a. $Q1S1D$ is generated by joining the variable ?*place* of the query $Q1S$, i.e., the object of the predicate *foaf:based_near* with the variable ?*country* of the query $Q1D$, i.e. the subject of the predicates *rdfs:label* and *dbpedia2:capital*. The joined variable ?*country* has been renamed by ?*place*, in the generated query $Q1S1D$. The execution of this query over a federation of SWDF and DBpedia produces 1056 results.

The generated query $Q1S1D$ can be recommended as a plausible federated query because for an external observer $Q1S$ and $Q1D$ are subqueries of $Q1S1D$, if we ignore the renaming of variables. In the same way, we can generate a more complex federated query such as the query $Q2S2D$ shown in Figure 2b. $Q2S2D$ is obtained by extending the query $Q2S$ from the log of SWDF with the query $Q2D$ from the log of DBpedia. The joining variable ?*sameAs* is renamed as ?*person* in $Q2S2D$. The objective of this paper is to show how we can generated plausible generated queries automatically. Currently, we cannot find a federated query log. This is normal, because a federated engine like FedX [15], for example, decomposes a federated query into sub-queries, and evaluates these sub-queries against relevant SPARQL endpoints. Therefore, a query log of a SPARQL endpoint does not know the federated query, it is only aware of a fragment of the federated query.

[1] `https://github.com/dice-group/feasible`

[2] Common prefixes are used and swc:<http://data.semanticweb.org/ns/swc/ontology#>, swrc:<http://swrc.ontoware.org/ontology#>,dbpedia:<http://dbpedia.org/resource/>, dbpedia2:<http://dbpedia.org/property/> and dbo:<http://dbpedia.org/ontology/>

```
SELECT * WHERE {
  SERVICE <http://swdf-2012>
    { ?obj rdf:type foaf:Organization .
      ?obj foaf:based_near ?place
  SERVICE <http://dbpedia-3.5.1>
    { ?place rdfs:label "United Kingdom"@en .
      ?place dbpedia2:capital ?capital .
      ?capital geo:lat ?lat .
      ?capital geo:long ?long }}}
 #results = 1 056
```

(a) $Q1S \bowtie Q1D$

```
SELECT * WHERE {
  SERVICE <http://swdf-2012> {
  swc:tim-finin rdf:type foaf:Person
      {swc:tim-finin foaf:name ?name1}
    UNION
      {swc:tim-finin rdfs:label ?name1}
    OPTIONAL
      {swc:tim-finin foaf:mbox_sha1sum ?mbox_sha1sum}
    OPTIONAL
      {swc:tim-finin foaf:homepage ?homepage}
    OPTIONAL
      {swc:tim-finin foaf:page ?page}
    OPTIONAL
      {swc:tim-finin owl:sameAs ?person
      SERVICE <http://dbpedia-3.5.1> {
        ?person skos:subject ?subject .
        ?person dbo:birthDate ?birth .
        ?person foaf:name ?name2 .
        ?person rdfs:comment ?description
        FILTER (lang(?description) = "en")}}
    OPTIONAL
      {swc:tim-finin rdfs:seeAlso ?seeAlso}
  }}
   #results = 178
```

(b) $Q2S \bowtie Q2D$

Fig. 2: Plausible federated queries generated from logs of SWDF 2012 and DBpedia 3.5.1 in Figure 1

We believe generating plausible federated queries over real datasets allows to leverage the full power of the LOD. Recommending real federated queries is challenging. This requires to explore a large number of queries to find joins among datasets. In addition, not all combinations of joinable queries generate a federated query that is semantically correct [10,5]. We propose PFed, a new approach to generate *plausible* and *correct* federated queries. PFed starts by pruning the queries logs by selecting only queries with joinable predicates reducing the search space. The contributions of the paper are:

- We define a new semantic summary for pruning query logs.
- We define an algorithm to exclude non joinable queries from logs.
- We propose an approach to generate plausible and correct federated queries using the pruned logs.
- We validate our approach using queries of SWDF 2012 and DBpedia 3.5.1.

This paper is organized as follows. Section 2 summarizes related works. Section 3 details PFed, an automatic generator of plausible federated queries. Section 4 presents our experimental results. Finally, conclusions and future work are outlined in Section 5.

2 Related Work

Feta [8] is a federated query tracker that computes Basic Graph Patterns from a federated log. It supposes the existence of a federated query log. In this work, we want to recommend federated queries rather than analyzing federated query logs.

Many efforts have been done to automatically generate SPARQL queries, either for individual dataset [4,12] or multiple datasets as Splodge [7] and FedBench [14]. Federated queries benchmarks have been proposed for evaluating

the performance of federated query engine. Existing benchmark rely either on hand-crafted queries or on automatically generated ones.

FedBench [14] rely on hand-crafted queries. The datasets of FedBench are real datasets preselected from the Linked Data Cloud, e.g. Life Science, Cross domain. FedBench is commonly used for the evaluation of federated query engines. FedBench is not designed to recommend plausible federated queries over a federation of SPARQL endpoints. LargeRDFBench [11] attempts to generate more realistic federated queries. The benchmark comprises a total of 32 queries for SPARQL endpoint federation. Queries are ranging from simple queries extracted from FedBench queries and large data queries created by the authors with the help of the expert domain. As FedBench, LargeRDFBench are designed for preselected datasets and queries are designed for specific domains and cannot be used for automatic generation of realistic federated queries.

Splodge [7] proposes heuristics for automatic query generation. Splodge generates only conjunctive queries of triple patterns, i.e., Basic Graph Patterns (BGP) with bound predicate, unbound subject and unbound object. Other SPARQL operators such as FILTER, OPTIONAL are not considered. However, recent analytical study of large SPARQL query logs [6] shows that 74.83% of studied queries have JOIN, FILTER and OPTIONAL and only 7.49% have JOIN alone (conjunctive queries). Consequently, the queries of Splodge cannot reflect the reality.

Existing approaches of automatic generation of federated queries do not reflect reality and hand-crafted federated queries are designed for specific datasets with the purpose to stress the performance of a federated query engine. Benchmarks are not designed for recommending plausible federated queries.

3 Generation of Plausible Federated Queries

Intuitively, for generating a plausible federated query over n datasets, we start by combing (joining) the query logs log_1 and log_2 of two datasets d_1 and d_2, each dataset is hosted by a SPARQL endpoint. We can distinguish different type of join combinations: subject-subject or object-subject leading to different query structures star-shaped, path-shaped, or hybrid queries [14]. Then, we generate new federated queries by joining the resulting queries and the log log_3 of the dataset d_3. We repeat the same process iteratively until processing the n query logs.

Processing the whole logs to generate federated queries means that any query from one log could be combined with at least a query from the another log. However, this not always the case, for instance, in our experimentation for some join combinations only approximately 25% of query log of SWDF can be joined with the log of DBpedia. Therefore, we need to prune the logs to keep only joinable queries, i.e., queries contain at least one joinable predicate.

The problem is given two logs log_1 and log_2, for a query $Q_1 \in log_1$ checks if it exists a query $Q_2 \in log_2$ where Q_1 is joinable with Q_2.

3.1 Finding joinable queries

To find joinable predicates, one can rely on the Vocabulary Of Interlinked Datasets VoID [3]. This vocabulary describes metadata about RDF datasets and the *linkset*. A *linkset* is a collection of RDF links between two datasets [3]. An RDF link is an RDF triple whose subject and object are described in different datasets. This corresponds to the joinable predicates in the example of the Figure 2. However, we cannot use VoID to detect joinable predicates because a large number of RDF datasets do not provide VoID [16], only 13.65% of datasets [4] (77/564) present a VoID description.

Another solution is to use the capabilities of data sources as defined in Hibiscus [13] to check the *possible* existence of matching. According to [13], the data summary of a source $d \in D$ is the set $CA(d)$ of all *capabilities* of that source.

Definition 1 (Capability). *Given a source d, a capability is a triple (p, SA(d, p), OA(d, p)), which contains (1) a predicate p in d, (2) the set $SA(d, p)$ of all distinct* subject authorities *of p in d and (3) the set $OA(d, p)$ of all distinct* object authorities *of p in d.*

The total number of capabilities of a source is equal to the number of distinct predicates in it. The definition of the authorities of a subject or an object relies on the analysis of the Unified Resource Identifier (URI) syntax. The URI syntax consists of a hierarchical sequence of components referred to as the *scheme*, *authority*, *path*, *query*, and fragment[5]. For example, the uri `<http://dbpedia.org/ontology/Plant>` contains a schema `"http"`, an authority `"dbpedia.org"` and a path `"ontology/Plant"`. To compute the set of capabilities for a source, the first two components (path, authority) are combined as the *authority* of the URI. Figure 3 presents a sample of the summary of SWDF 2012 and DBpedia 3.5.1. For instance, in Figure 3a, the first capability of SWDF data source is the predicate *foaf:based_near*, its subject authority is `<http://data.semanticweb.org>` and its object authorities are `<http://dbpedia.org>`, `<http://www.w3.org>`, `<http://sws.geonames.org>`, and `<http://data.semanticweb.org>`.

Hibiscus data summary allows to prune the query logs only if many predicates have different subjects or objects authority. However, this not always the case, especially for the subject authority. For instance, the majority of subjects of DBpedia have the authority `<http://dbpedia.org>`, only six predicates out of 39672 predicates of DBpedia 3.5.1 do not have `<http://dbpedia.org>` as a subject authority. Therefore, if a query Q_1 in SWDF query log is joinable with a query Q_2 in DBpedia query log on the subject authority `<http://dbpedia.org>`, then Q_1 will be joinable with a large number queries in the log of DBpedia. Therefore, for query logs of SWDF and DBpedia, Hibiscus will prune mostly queries with unbounded predicates.

We need to use the semantic of subjects and objects for finding joinable predicates. Intuitively, a subject or an object from one dataset could be joinable with

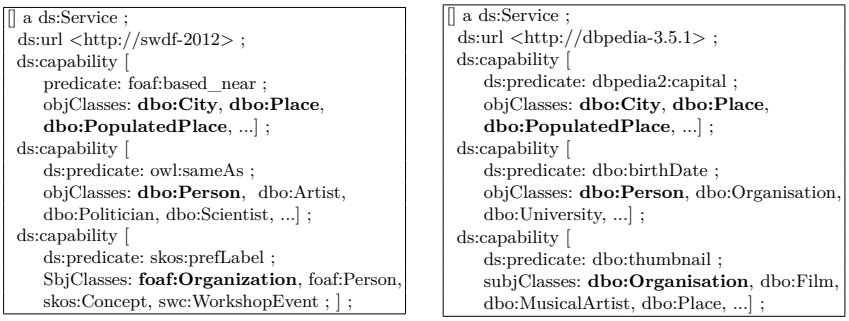
[3] `https://www.w3.org/TR/void`

[4] `http://sparqles.ai.wu.ac.at/`

[5] URI Syntax Components: https://tools.ietf.org/pdf/rfc3986.pdf

(a) SWDF data summary

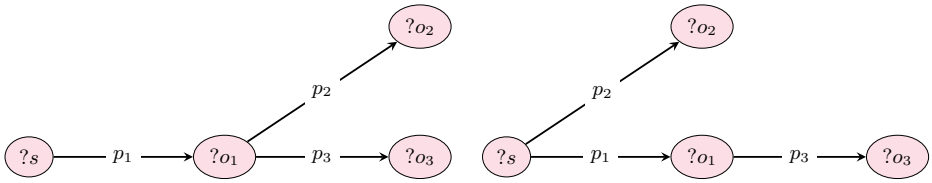
(b) DBpedia data summary



(c) SWDF classes summary

(d) DBpedia classes summary

Fig. 3: Sample of Hibiscus Summaries and Classes Summaries of logs of SWDF 2012 and DBpedia 3.5.1



(a) p_1, $p_2 \in Q_1$, $Q_1 \in log_1$ and $p_3 \in Q_2$, $Q_2 \in log_2$

(b) $p_1 \in Q_1$, $Q_1 \in log_1$ and p_2, $p_3 \in Q_2$, $Q_2 \in log_2$

Fig. 4: Possible structures for hybrid federated queries

a subject or object from another dataset, if they share some common types. More precisely, we define a new summary called *Class summary*. A class summary is a set of capabilities where a capability consists of a predicate, the set SC(d, p) of classes of its subject and the set OC(d, p) of classes of its object. The classes are computed using SPARQL queries. For instance, to compute the object classes of the predicate *foaf:based_near*, we execute the following SPARQL federated query:

```
SELECT DISTINCT ?type WHERE {
      SERVICE <http://swdf-2012> {
      ?conf foaf:based_near ?x .
       SERVICE <http://dbpedia-3.5.1>
          {?x rdf:type ?type}}
}
```

If we want to find joinable predicates for path query from SWDF to DBPedia, we look for the type of the object of the predicate in DBPedia dataset. Figures 3c and 3d present Classes summary for SWDF and DBpedia, respectively.

We use only the direct classes of subjects and objects to find common classes. We do not use inferences to find common classes because schemas information are not always available [9].

3.2 Pruning query logs

Based on data summaries and classes summaries, we can prune the logs of corresponding datasets by retaining only joinable queries. The pruning depends on the structure of generated federated queries. Let D be a set of distinct data sources, $d_1, d_2 \in D$. Let S_1, SC_1, S_2, SC_2 be the summaries of d_1 and d_2, log_1 and log_2 are the query log of d_1 and d_2, respectively. For two queries $Q_1 \in log_1$ and $Q_2 \in log_2$ with $tp_1 = (s_1, p_1, o_1) \in Q_1$ and $tp_2 = (s_2, p_2, o_2) \in Q_2$, we say that Q_1 and Q_2 are joinable if p_1 and p_2 are *joinable path* or *joinable star*.

Definition 2 (Joinable Path). *joinablePath*(p_1, p_2) = *true, if* $OA(d_1, p_1) \cap SA(d_2, p_2) \neq \emptyset$ *and* $OC(d_1, p_1) \cap SC(d_2, p_2) \neq \emptyset$.

Definition 3 (Joinable Star). *joinableStar*(p_1, p_2) = *true, if* $SA(d_1, p_1) \cap SA(d_2, p_2) \neq \emptyset$ *and* $SC(d_1, p_1) \cap SC(d_2, p_2) \neq \emptyset$.

Algorithm 1: Algorithm for Excluding non PATH Joinable queries between log1 and log2

Input: $log_1, log_2, S_1, S_2, SC_1, SC_2$ ▷ query logs and summaries
Output: $PJlog_1, PJlog_2$ ▷ Joinable Path Queries of log_1 and log_2

```
1  Function PJDetection(F):
2      PJlog1 ⟵ ∅;
3      PJlog2 ⟵ ∅;
4      foreach Q1 ∈ log1 do
5          foreach Q2 ∈ log2 do
6              let OA1 = S1.OAuth.Predicate(Q1)
7              let SA2 = S2.SAuth.Predicate(Q2)
8              if OA1 ∩ SA2 ≠ ∅ then
9                  let OC1 = SC1.OClasses.Predicate(Q1)
10                 let SC2 = SC2.SClasses.Predicate(Q2)
11                 if OC1 ∩ SC2 ≠ ∅ then
12                     PJlog1 ⟵ PJlog1 ∪ Q1
13                     PJlog2 ⟵ PJlog2 ∪ Q2
14                 end
15             end
16         end
17     end
18     return PJlog1, PJlog2;
19 End Function
```

The hybrid join pattern is built as a mix of a path join pattern and a star join pattern. Figure 4 presents possible structures of hybrid federated queries. The query generated in Figure 4a is built from the path query of $p_1 \in Q_1$, $Q_1 \in log_1$ and $p_3 \in Q_2$, $Q_2 \in log_2$. The query generated in Figure 4b built from the star query of $p_1 \in Q_1$, $Q_1 \in log_1$ and $p_2 \in Q_2$, $Q_2 \in log_2$.

The algorithm 1 presents how to prune query logs to construct path-shaped federated queries. To illustrate, consider a sample of query logs of SWDF 2012 and dbpedia-3.5.1 in figures 1a and 1b, respectively.

After the execution of the algorithm, only $Q1S$, $Q2S$ of SWDF and $Q1D$ and $Q2D$ of DBpedia will be preserved, because the intersection between the object authorities of *based_near* $\in Q1S$ and the subject authority of *dbpedia2:capital* $\in Q1D$ is not empty since they have the authority `<http://dbpedia.org>` and the type *dbo:city* in common. As well, the object authorities of *owl:sameAs* $\in Q2S$ and the subject authority of *dbo:birthDate* $\in Q2D$ have the authority *<http://dbpedia.org>* and the type *dbp:Person* in common. We exclude $Q3S$ because it cannot be joined with any query from dbpedia, *i.e.* no predicate in dbpedia has `<http://data.semanticweb.org>` as subject authority. We also eliminate $Q3D$ because the capability of unbound predicate is undefined.

For star-shaped federated queries (subject-subject), we have to modify the condition in line 8 of algorithm 1 by $(SA_1 \cap SA_2 \neq \varnothing)$ and the condition in line 11 by $(SC_1 \cap SC_2 \neq \varnothing)$. In this case, we keep queries $Q4S$ and $Q4D$.

The subject authorities of *skos:prefLabel* $\in Q4S$ and of *dbo:thumbnail* $\in Q4D$ have the authority `<http://dbpedia.org>` and the type *foaf:Organization* in common.

```
SELECT * WHERE {
  SERVICE <http://swdf-2012>
    {?obj foaf:based_near ?place .
  SERVICE <http://dbpedia-3.5.1>
    {?place dbpedia2: capital ?capital }}}
```

(a) Path query by joining a triple pattern from $Q1S$ and a triple pattern from $Q1D$

```
SELECT * WHERE {
  SERVICE <http://swdf-2012>
    {?x rdfs:prefLabel ?o1 .
  SERVICE <http://dbpedia-3.5.1>
    {?x dbo:thumbnail ?o2 }}}
```

(b) Star query by joining a triple from $QS4$ and a triple pattern from $Q4D$

Fig. 5: Minimal federated queries generated from pruned logs of SWDF and DBpedia in Figure 3

3.3 Building plausible federated queries

The construction of plausible federated queries follows the following process:
- Use pruned logs to build minimal federated queries.
- Use minimal federated queries to build plausible federated queries.

We start by generating minimal federated queries PFED_{min}. A minimal federated contains one triple from log_1 and one triple from log_2.

Minimal federated queries $\mathbf{PFed_{min}}$ is a basic federated query, it is composed of two triples patterns and defined as follows.

Definition 4 ($\mathbf{PFed_{min}}$). *Let log_1, log_2 be two pruned logs of queries of datasets d_1 and d_2, respectively. $p_1 \in log_1$ and $p_2 \in log_2$ are two predicates.*

PFED_{min} = *{ $?s, p_1, ?x$. $?x, p_2, ?o$ / $joinablePath(p_1, p_2) = true$}*
or {$?s, p_1, ?o_1$. $?s, p_2, ?o_2$ / $joinableStar(p_1, p_2) = true$ }

In order to construct a path (start) join, we substitute the object (subject) of p_1 and the subject of p_2 by the same value as given in the table 1.

tp_1 object	$tp2$ subject	substitution value
?x	?y	?x
?x	a	a
a	?x	a
a	b	null

Table 1: All substitution values possible to create path join. $?x$, $?y$ are variables and a, b are constants (URIs or literals)

Figure 5a presents a minimal path-shaped federated query between *foaf:based_near* $\in Q1S$ and *dbpedia2:capital* $\in Q1D$ in Figure 3. Figure 5b presents a minimal star-shaped federated query between *skos:prefLabel* $\in Q4S$ and *dbo:thumbnail* $\in Q4D$.

We keep only PFED_{min} queries that produce results to generate plausible federated queries.

```
SELECT * WHERE {
    ?s1 p1 ?o1 .
    ?s1 p2 ?o2  }
```

(a) Q_1 from log endpoint1

```
SELECT * WHERE {
    ?s3 p3 ?o3
    OPTIONAL { ?o2 p4 ?o3 }}
```

(b) Q_2 from log endpoint2

```
SELECT * WHERE {
    SERVICE <endpoint1>
    { ?s1 p1 ?o1 .
      ?s1 p2 ?o2
    SERVICE <endpoint2>          :P'
    { ?s3 p3 ?o3                 :P1
      OPTIONAL { ?o2 p4 ?o3 }}}} :P2
```

(c) $Q_1 \bowtie Q_2$

Fig. 6: A non well designed federated query

Plausible Federated queries PFed The construction of Q_{PFED} is tricky, if the original queries contain OPTIONAL operator. We have to construct only correct plausible federated query. A plausible federated query is *correct* if it is well designed [10] and service-safeness [5].

Definition 5 (Well designed[10]). *A graph pattern P is well designed if for every occurrence of a sub-pattern P' = (P1 OPT P2) of P and for every variable ?X occurring in P, the following condition holds:*

if ?X occurs both inside P2 and outside P', then it also occurs in P1.

The federated query in Figure 6c is not well designed because the variable ?o2 occurs in P2 and outside the P' (i.e. clause SERVICE <dataset2>), but it not occurs in P1.

The service-safeness provides condition that ensures that a SPARQL query containing SERVICE operator can be safely evaluated. Our generated queries ensure service-safeness because each SERVICE clause has only bounded service, i.e., during the construction the URI of the SPARQL endpoints are known.

The main issue is to build well designed queries. A correct plausible federated query is constructed as follows. Let log_1, log_2 be two pruned query logs, $Q_1 \in log_1$, $Q_2 \in log_2$, Q_1 and Q_2 contribute to the construction of a minimal query, then:

- If Q_1 and Q_2 are conjunctive queries (a.k.a BGPs) then $Q_{\text{PFED}} = Q_1 \bowtie Q_2$, Q_{PFED} is a simple concatenation of queries (Q_1 . Q_2), as in figure 2a, $Q1S1D = Q1S \bowtie Q1D$.
- If Q_1 contains binary operators like UNION or OPTIONAL, we distinct two cases:
  - If a joinable predicate is outside binary clauses of Q_1, we add Q_2 in the BGP part of Q_1.
  - If a joinable predicate of Q_1 is inside the UNION or OPTIONAL clauses, we append Q_2 inside this clause after the substitution of the join variables (subject or object of the triple) according to table 1. Figure 2b presents a federated where the joinable predicate *owl:sameAs* of $Q2S$ is inside the OPTIONAL Clause.
- If a joinable predicate of Q_2 is inside an OPTIONAL clause, we make sure to not generate non well designed queries like query shown in 6c.

4 Evaluation

We want to answer empirically the following questions: Does Hibiscus summary prune non joinable predicates? Does Classes summary effective? Does PFED produce "realistic" queries compared to automatic query generation benchmarks? Does PFED generate correct plausible federated queries?

All data, codes, and generated query are available at the project web page [6].

4.1 Experimental Setup

Dataset and Queries: We use SWDF 2012 and DBPedia 3.5.1 datasets and clean queries of Feasible [7]. Table 2 reports statistics about the datasets and query logs. It is strange that the query log of SWDF contains more predicates than the original dataset hosted at the SPARQL endpoint. Some queries in the logs use predicates that are not defined in the dataset. Using DBpedia to generate plausible federated queries is challenging because DBpedia dataset has a high number of predicates and the log of DBpedia has a high number of queries. We use only SELECT queries to construct plausible federated queries.

dataset	|triples|	|dataset predicates|	|original log|	|SELECT queries|	|log predicates|
SWDF	242 256	170	64 030	37 592	201
DBPedia	232 542 405	39 672	217 812	127 812	247

Table 2: Real Datasets and real logs

[6] `https://github.com/GDD-Nantes/PFed`
[7] https://github.com/dice-group/feasible

Approaches: We can compare our approach only with Splodge, because only Splodge is automatic federated queries benchmark. However, we can compare only with minimal queries since Splodge generates only conjunctive queries (without OPTIONAL, UNION). We modify the code of Splodge[8] to generate all possible queries instead of stopping after the generation of the first couple of joinable predicates.

4.2 Experimental Results

Does Hibiscus summary prune non joinable predicates ? Table 3 presents the results of pruning using data summaries of Hibiscus. As we can see, the reduction is 62.75% for SWDF query log for path-shaped queries and by 42.82% for star-shaped. The reduction if only 2.15% of DBpedia log for both path-shaped queries and star-shaped generation. This reduction is not significant because most of predicates in DBPedia has the authority <http://dbpedia.org>.

	path-shaped			star-shaped		
dataset	|predicate joinable |	|pruned log|	% reduce	|predicate joinable |	|pruned log|	% reduce
SWDF	6	14 003	62.75	3	21 475	42.82
DBPedia	229	125 070	2.15	229	125 070	2.15

Table 3: Logs pruning using Hibiscus summaries

Does Classes summary effective? We use Classes summary to prune the log returned by Hibiscus. Classes summary is effective because it increases considerably the percentage of pruned queries. The reduction is impressive for DBPedia from 2.15% with Hibiscus to 71.42% with Classes summary as shown in table 4.

	path-shaped			star-shaped		
dataset	|predicate joinable |	|pruned log|	% reduce	|predicate joinable |	|pruned log|	% reduce
SWDF	3	9 355	75.12	3	21 495	42.82
DBPedia	139	36 522	71.42	83	36 449	71.48

Table 4: Logs pruning using Hibiscus & Classes summaries

Does PFED *produce "realistic" queries compared to automatic query generation benchmarks?* We run Splodge on both SWDF and DBpedia. As parameters, we used a low minimum selectivity to get at least one result. We also didn't use a max selectivity in order to get as much queries as possible.

[8] We do not use the original code of Splodge because it does not implement Star-shaped queries, we use the code at `https://github.com/Institute-Web-Science-and-Technologies/splodge`

Query Structure	|queries|	|predicates SWDF|	|predicates DBpedia |
Path SWDF to DBpedia	1 300	6	1 108
Path DBpedia to SWDF	170	3	98
Star	63	3	38

Table 5: Statistics of minimal queries generated by Splodge using SWDF and DBpedia

Path-shaped queries SWDF to DBpedia			Star-shaped queries		
all generated	with result	%	all generated	with result	%
397	77	19.39	249	20	8.03

Table 6: Statistics of minimal queries generated by PFED using Hibiscus & Classes summaries

Table 5 presents minimal queries generated by Splodge. Table 6 shows that from 3 joinable predicates of SWDF and 139 joinable predicates of DBPedia (see table 4), PFED generates only 77 minimal path-shaped queries with non empty result set. Whereas, Splodge generates 1300 queries, some of these queries are not significant as the query displayed in Figure 7. PFED generates relatively less queries, 20 PFED_{min} star-shaped queries with non empty result set based on 3 joinable predicates of SWDF and 83 joinable predicates of DBPedia. PFED relies on query logs to generate plausible queries.

Does PFED *generate correct plausible federated queries ?* Due to the size of the pruned logs, we can generate a large number of plausible federated queries. In our experimentation, we focus on the generation of path-shaped between *foaf:based_near* from SWDF and *dbpedia2:capital* from DBPedia because *foaf:based_near* predicate is used to link SWDF and DBpedia datasets. The pruned SWDF query log contains 2 866 queries that contains *foaf:based_near*. Many of these queries have the same structure but with different literals and variables. Therefore, instead of producing $2866 \times 14 = 40124$ queries where 14 is the number of queries that contains *dbpedia2:capital* in pruned DBpedia log, we define patterns for *foaf:based_near* queries. We differentiate 9 patterns for *foaf:based_near* queries and we generate 24 queries. All generated queries are executed correctly and 19 of these queries have non empty results set (see table 7).

```
SELECT ?v0 ?v2 WHERE {
    SERVICE <http://swdf-2012>
    { ?v0 foaf:based_near ?v1
    SERVICE <http://dbpedia.org> {
        ?v1 dbpedia2:image2Caption ?v2 . }}}
```

Fig. 7: A minimal federated query generated by Splodge

	p_1	$\|p_1\|$	p_2	$\|p_2\|$	|PFED|	|with result|	%
PFED path	foaf:based_near	9	dbpedia2:capital	5	24	19	79.17
PFED star	skos:prefLabel	3	dbo:thumbnail	14	42	14	33.33

Table 7: PFED path and star, $p_1 \in$ SWDF and $p_2 \in$ DBPedia

We generate star-shaped plausible federated queries based on *skos:prefLable* from SWDF and *dbpedia:thumbnail* from DBPedia (see table 7). The 42 generated queries are executed correctly and 28 of these queries produce results.

5 Conclusion and Future Work

We presented PFED an approach for automatic generation of plausible federated queries based on real query logs. PFED starts by pruning the logs to exclude non joinable queries using two data summaries. The first one is based on the authorities and the second is based on the type of subjects and objects of predicates. Experimentations with real query logs of SWDF and DBpedia demonstrate that PFED is able to prune considerably the logs and generate correct plausible federated queries.

As future work, we would like to experiment PFED with more real query logs and produce plausible federated queries over a large number of SPARQL endpoints. Finally, we plan to extend PFED with statistical information to generate only queries that return results.

Acknowledgement

This work is part of the multidisciplinary project SEDELA, funded by CominLabs, that brings together three laboratories: LS2N, CREAD and Lab-STICC.

References

1. Acosta, M., Vidal, M., Lampo, T., Castillo, J., Ruckhaus, E.: ANAPSID: an adaptive query processing engine for SPARQL endpoints. In: International Semantic Web Conference. Lecture Notes in Computer Science, vol. 7031, pp. 18–34. Springer (2011)
2. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art. IEEE transactions on knowledge and data engineering 17(6), 734–749 (2005)
3. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing linked datasets. In: LDOW (2009)
4. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: The International Semantic Web Conference. pp. 197–212 (2014)
5. Arenas, M., Pérez, J.: Federation and navigation in sparql 1.1. In: Reasoning Web International Summer School. pp. 78–111. Springer (2012)

6. Bonifati, A., Martens, W., Timm, T.: An analytical study of large SPARQL query logs. PVLDB 11(2), 149–161 (2017), http://www.vldb.org/pvldb/vol11/p149-bonifati.pdf
7. Görlitz, O., Thimm, M., Staab, S.: SPLODGE: systematic generation of SPARQL benchmark queries for linked open data. In: The Semantic Web - ISWC 2012 - 11th International Semantic Web Conference, Boston, MA, USA, November 11-15, 2012, Proceedings, Part I. pp. 116–132 (2012), https://doi.org/10.1007/978-3-642-35176-1_8
8. Nassopoulos, G., Serrano-Alvarado, P., Molli, P., Desmontils, E.: FETA: Federated QuEry TrAcking for Linked Data. In: International Conference on Database and Expert Systems Applications (DEXA). p. 0. No. 9828 in Lecture Notes in Computer Science (Sep 2016)
9. Neumann, T., Moerkotte, G.: Characteristic sets: Accurate cardinality estimation for rdf queries with multiple joins. In: Data Engineering (ICDE), 2011 IEEE 27th International Conference on. pp. 984–994. IEEE (2011)
10. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. In: International semantic web conference. pp. 30–43. Springer (2006)
11. Saleem, M., Hasnainb, A., Ngonga Ngomo, A.C.: LargeRDFBench: A billion triples benchmark for sparql endpoint federation. In: Journal of Web Semantics (JWS) (2017), https://svn.aksw.org/papers/2017/LargeRDFBench_JWS/public.pdf
12. Saleem, M., Mehmood, Q., Ngomo, A.C.N.: Feasible: A feature-based sparql benchmark generation framework. In: International Semantic Web Conference. pp. 52–69. Springer (2015)
13. Saleem, M., Ngomo, A.C.N.: Hibiscus: Hypergraph-based source selection for sparql endpoint federation. In: European Semantic Web Conference. pp. 176–191. Springer (2014)
14. Schmidt, M., Görlitz, O., Haase, P., Ladwig, G., Schwarte, A., Tran, T.: Fedbench: A benchmark suite for federated semantic data query processing. In: International Semantic Web Conference. pp. 585–600 (2011), https://doi.org/10.1007/978-3-642-25073-6_37
15. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: Optimization techniques for federated query processing on linked data. In: International Semantic Web Conference. pp. 601–616. Springer (2011)
16. Vandenbussche, P.Y., Umbrich, J., Matteis, L., Hogan, A., Buil-Aranda, C.: Sparqles: Monitoring public sparql endpoints. Semantic Web 8(6), 1049–1065 (2017)