PFed: Recommending Plausible Federated SPARQL Queries

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

PFed: Recommending Plausible Federated SPARQL Queries

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

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

Abstract. Federated SPARQL queries allow to query multiple interlinked datasets hosted by remote SPARQL endpoints. However, finding 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 real 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 from a given log, PFed stretches the query with real queries from different logs. To prune the research space, PFed proposes semantic summary to prune the query logs. Experimental results with real logs of DBpedia and SWDF demonstrate that PFed is able to prune drastically the logs and recommend plausible federated queries.

Keywords: Semantic Web, 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. However, finding 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 original approach to recommend federated queries for end-users. Instead of using datasets to recommend federated queries, PFed recommends federated queries using 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 given log and we stretch this query with real queries from other existing query logs. The main advantage of using real logs rather than using datasets is to produce *plausible* federated queries, i.e. queries that generated by combining real queries. This is useful, especially for data portal owners who can recommend federated queries for end-users. Imagine a data portal such as

Sage [1], or LodLaundromat [2] hosting thousands of linked datasets. The portal owner can see that some users are looking for information about "United Kingdom" in DBpedia, others are looking for conferences in SWDF dataset. Using PFed, the portal owner can suggest to extended conferences with information about country.

To illustrate, consider queries extracted from real SPARQL query logs of SWDF (SWDF 2012) and DBpedia (DBpedia 3.5.1) [3] presented in Figure 1.

```
Q1S: SELECT * WHERE {
       ?inst rdf:type ?dClass .
       ?inst foaf:based_near ?place
 }#results:5025

Q3S: SELECT * WHERE {
       {?paper swrc:author ?author}
       UNION {?paper foaf:maker ?author}
       OPTIONAL {?paper swrc:abstract ?abstract}
}#results:21649
```

(a) SWDF query log

```
Q1D:SELECT * WHERE {
       ?country rdfs:label "United Kingdom"@en .
       ?country dbp:capital ?capital .
       ?capital geo:lat ?lat .
       ?capital geo:long ?long
 }#results: 4

Q3D: SELECT * WHERE {
       {dbpedia:Paris ?property ?hasValue}
     UNION
       {?isValueOf ?property dbpedia:Paris}
}#results:41482
```

(b) DBpedia query log

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

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 2. $Q1S1D$ 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 1388 results.

The generated query $Q1S1D$ can be recommended as a plausible federated query. 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 $Q2S2D$.

Recommending plausible federated queries is challenging because the size of logs. The log of DBpedia contains 217 812 queries, and the log of SWDF contains 64 030 queries [12]. To overcome this problem, we propose a semantic summary that allows to reduce drastically the size of logs by excluding non joinable queries. The main contributions of the paper are:

- a new semantic summary for pruning query logs.
- an algorithm to exclude non joinable queries from logs.

[1] http://sage.univ-nantes.fr

[2] http://lodlaundromat.org/

[3] All information about logs, and prefixes are available at the project site: `https://github.com/GDD-Nantes/PFed`

```
SELECT * WHERE {
  SERVICE <http://swdf-2012>
    { ?inst rdf:type ?dClass .
      ?inst foaf:based_near ?place
  SERVICE <http://dbpedia-3.5.1>
    { ?place rdfs:label "United Kingdom"@en .
      ?place dbp:capital ?capital .
      ?capital geo:lat ?lat .
      ?capital geo:long ?long }}}
 #results = 1388
```

(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 query generated from logs of SWDF and DBpedia in Figure 1

- an algorithm for generating plausible federated queries using the pruned logs.
- an experimentation using real queries logs of SWDF 2012 and DBpedia 3.5.1.

This paper is organized as follows. Section 2 summarizes related works. Section 3 details PFED approach and algorithms. Section 4 presents our experimental results. Finally, conclusions and future work are outlined in Section 5.

2 Related Work

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. 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 build and recommend federated queries rather than analyzing federated query logs.

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 propose to start by combing (joining) the query logs log_1 and log_2 of two datasets d_1 and d_2, respectively. 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.

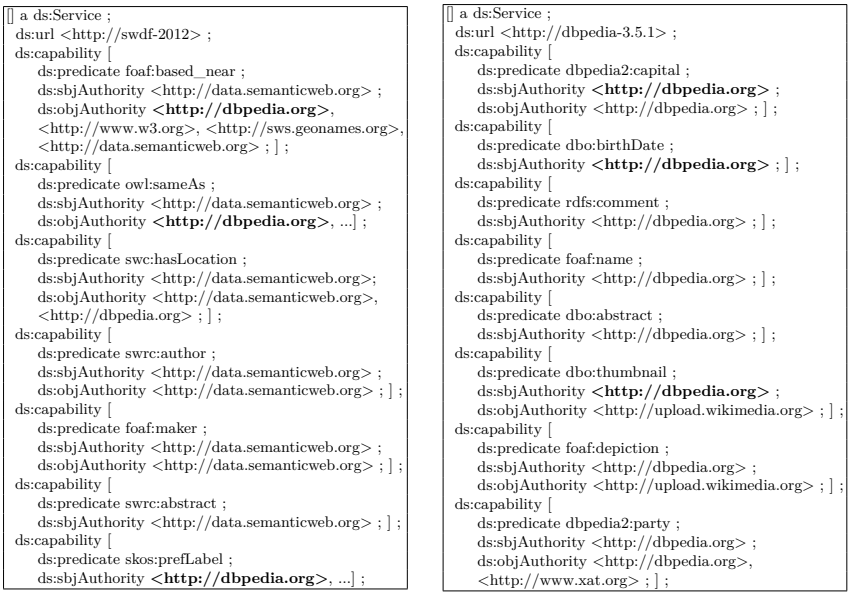
In the following, for simplicity, we restrict our discussion to the case of two real query logs. Given two queries Q_1 and Q_2 belong to different query logs, we want to build a plausible federated query FQ. We call FQ a *p*lausible federated query because it is composed of two real queries. Our intuition is FQ is more likely to be a real query than a synthetic one.

3.1 Datasets capabilities

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]. 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 [4]. 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 [5] (77/564) present a VoID description.

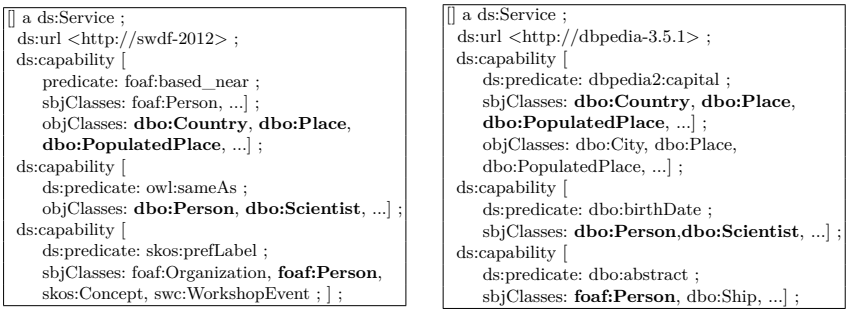
Another solution is to use the capabilities of data sources as defined in Hibiscus [13] to check the *possible* existence of matching. According to [13], the data summary of a source $d \in D$ is the set $CA(d)$ of all *capabilities* of that source. In Hibiscus, this summary is used to remove endpoints during the source selection during federated query processing

[4] `https://www.w3.org/TR/void`
[5] `http://sparqles.ai.wu.ac.at/`



(a) SWDF data summary

(b) DBpedia data summary



(c) SWDF classes summary

(d) DBpedia classes summary

Fig. 3: Sample of authorities and classes summaries of logs of SWDF and DBpedia

Definition 1 (Authority Capability). *Given a source d, an authority 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[6]. 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>`.

Authority 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 of queries in the log of DBpedia. Therefore, for query logs of SWDF and DBpedia, authority summary will prune mostly queries with unbounded predicates.

To further prune the log, we define new data summary that considers semantic of subjects and objects for finding joinable predicates. Intuitively, a subject or an object from one dataset could be joinable with 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 classes capabilities.

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

Classes capabilities can be computed using SPARQL queries. But since entities are reused across datasets, types of the subjects and objects for predicates maybe not defined locally. Therefore, we need to perform a SPARQL federated query to compute classes capabilities. We use only the direct classes of subjects and objects to find common classes, we do not use inferences because schemas information are not always available [9], and we restrict the computation to

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

```
SELECT DISTINCT ?type
 WHERE {
      ?s foaf:based_near ?o .
      FILTER isURI(?s)
      ?s rdf:type ?type
}
```

```
SELECT DISTINCT ?type
WHERE {
      ?s foaf:based_near ?o .
      FILTER isURI(?o)
      ?o rdf:type ?type
}
```

(a) Retrieving types of *foaf:based_near* from SWFD

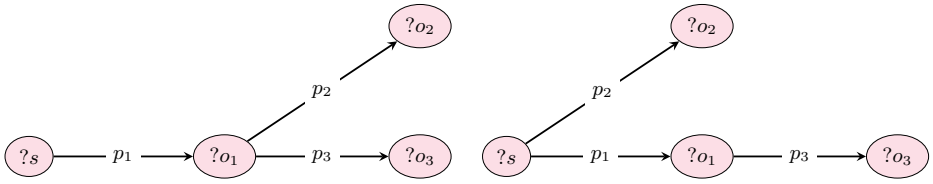
```
SELECT DISTINCT ?type WHERE {
   SERVICE<http://swdf-2012> {
     ?s foaf:based_near ?o .
     FILTER isURI(?s)
     FILTER regex(STR(?s), 'http://dbpedia.org')
     SERVICE<http://dbpedia3.5.1> {
        ?s rdf:type ?type
     }}
}
```

```
SELECT DISTINCT ?type WHERE {
     SERVICE<http://swdf-2012> {
     ?s foaf:based_near ?o .
     FILTER isURI(?o)
     FILTER regex(STR(?o), 'http://dbpedia.org')
     SERVICE<http://dbpedia3.5.1> {
        ?o rdf:type ?type
     }
}
```

(b) Retrieving types for *foaf:based_near* from DBpedia

Fig. 4: Class Capability for *foaf:based_near* predicate in SWDF

only used datasets. For instance, to compute the object classes of the predicate *foaf:based_near*, we rely only on SWDF and DBpedia. We define the following queries: Figures 3c and 3d present classes summaries for SWDF and DBpedia, respectively.



(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. 5: Possible structures for hybrid federated queries

3.2 Pruning query logs

Based on authorities summaries and classes summaries, we can prune the logs of corresponding datasets by retaining only joinable queries.

Definition 3 (Joinable queries). *Let D be a set of distinct data sources, $d_1, d_2 \in D$. Let log_1 and log_2 are the real 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 have a* predicate joinable path *or* predicate joinable star.

Definition 4 (Predicate 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 5 (Predicate Joinable Star). *joinableStar*(p_1, p_2) = *true, if* $SA(d_1, p_1) \cap SA(d_2, p_2) \neq \emptyset$ *and* $SC(d_1, p_1) \cap SC(d_2, p_2) \neq \emptyset$.

The hybrid join pattern is built as a mix of a path join pattern and a star join pattern. Figure 5 presents possible structures of hybrid federated queries. The query generated in Figure 5a 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 5b built from the star query of $p_1 \in Q_1$, $Q_1 \in log_1$ and $p_2 \in Q_2$, $Q_2 \in log_2$.

Algorithm 1: Joinable predicates

Input: $AS1, CS1, AS2, CS2$ ▷ Authorities and classes summaries for the two datasets
Output: $JPred$ ▷ Set of joinable predicates

```
1  Function JoinPred(AS1, CS1, AS2, CS2):
2      JPred ⟵ ∅;
3      foreach cap1 ∈ AS1 do
4          foreach cap2 ∈ AS2 do
5              if AS1.objAuthority(cap1) ∩ AS2.sbjAuthority(cap2) ≠ ∅ then
6                  if CS1.objClasses(cap1) ∩ CS2.sbjClasses(cap2) ≠ ∅ then
7                      JPred ⟵ JPred ∪ (cap1.predicate, cap2.predicate);
8                  end
9              end
10         end
11     end
12     return JPred;
13 End Function
```

The objective now is to prune query logs and conserve only joinable queries. First, the algorithm 1 uses summaries to conserve predicate joinable (predicate joinable path), then the algorithm 2 excludes non joinable queries from logs. For logs in Figure 3, the algorithm 1 keeps the couple (*foaf:based_near*,*dbpedia2:capital*) because they share *http://dbpedia.org* as object and subject authority, respectively, and they share *Country*, *Place* and *PopulatedPlace* as object and subject classes, respectively.

To compute predicate joinable star, we only need to modify conditions in lines 5-6 of the algorithm 1 to compare subjects parts of both capabilities. With this modification, the algorithm will keep the couple (*skos:prefLabel*,*dbo:abstract*) as they share same authorities and classes as subjects. The algorithm 1 can be iteratively called to compute predicate joinable path or star for more than two datasets.

We use the result of the algorithm 1 to exclude non joinable queries as shown in the algorithm2. After the execution of the algorithm 2 for joinable path, $Q1S$

Algorithm 2: Joinable queries

Input: $log_1, log_2, JPred$ ▷ Logs of both dataset and the set of corresponding joinable predicates
Output: $feds$ ▷ Set of federated queries

```
1  Function GenFed(log1, log2, JPred):
2      feds ⟵ ∅;
3      foreach Q1 ∈ log1 do
4          foreach Q2 ∈ log2 do
5              if ∃(p1, p2)|p1 ∈ Q1, p2 ∈ Q2 ∧ (p1, p2) ∈ JPred then
6                  feds ⟵ feds ∪ (Q1, Q2)
7              end
8          end
9      end
10     return feds;
11 End Function
```

of SWDF and $Q2D$ of DBpedia will be preserved, because they have the joinable predicates (*foaf:based_near*,*dbpedia2:capital*) as shown previously. 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.

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

3.3 Building plausible federated queries

We rely on the results of the algorithm 2 to build plausible federated queries. For sake of simplification, we start by illustrating the generating of minimal federated queries PFED_{min}. A minimal federated contains one triple from log_1 and one triple from log_2.

In order to construct a path (star) 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 6a presents a minimal path-shaped federated query between *foaf:based_near* $\in Q1S$ and *dbpedia2:capital* $\in Q1D$ in Figure 3. Figure 6b presents a minimal star-shaped federated query between *skos:prefLabel* $\in Q4S$ and *dbo:thumbnail* $\in Q4D$.

PFED$_{min}$ are not required to generate plausible federated queries. But they can help to reduce the number of potential joinable predicates by only keeping PFED$_{min}$ producing results. They can also be used to navigate through datasets.

```
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. 7: A non well designed federated query

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 6 (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 7c 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 to avoid cartesian products as illustrated in Figure 7c. If Q_2 does not have a mapping for ?o2, a result will still produced. To avoid this problem, we define the following strategy:

- 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 2, $Q1S1D = Q1S \bowtie Q1D$.
- 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.
- 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 7c.

4 Evaluation

The objective of the evaluation is to answer empirically the following questions: Do authorities summaries prune non joinable predicates? Do classes summaries prune further non joinable predicates? Does PFED able to generate plausible federated queries?

All data, codes, and generated query are available at the project web page [7].

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

4.1 Experimental Setup

Dataset and Queries: We use SWDF 2012 and DBPedia 3.5.1 datasets and clean queries of Feasible [8]. We use only SELECT queries to construct plausible federated queries. Table 2 reports statistics about the datasets and query

[7] `https://github.com/GDD-Nantes/PFed`

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

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. As they appear inside OPTIONAL or UNION, they do not stop queries from returning results. 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.

4.2 Experimental Results

	path-shaped			star-shaped		
dataset	|predicate joinable |	|pruned log|	% reduce	|predicate joinable |	|pruned log|	% reduce
SWDF	6	14 003	62.75	3	21 495	42.82
DBPedia	230	125 078	2.15	229	125 070	2.15

Table 3: Logs pruning using authorities summaries

Do authorities summaries prune non joinable predicates ? Table 3 presents the results of pruning using authorities summaries. As we can see, the reduction is 62.75% for SWDF query log for path-shaped queries (all path refers to path from SWDF to DBpedia) and by 42.82% for star-shaped. The reduction is only 2.15% for 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	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 authorities and classes summaries

Do classes summaries prune further non joinable predicates? We now use our classes summaries on top of authorities summaries. We observe in Table 4 that the sizes of logs are reduced. The reduction is impressive for DBpedia, it is about 72 %. Therefore, classes summaries are affective for pruning non joinable queries.

We observe also an important reduction in the number of minimal federated queries PFED$_{min}$ (Table 5). This reduction is important as each PFED$_{min}$ contributes to many federated queries.

With Authorities		With authorities and classes	
path-shaped	star-shaped	path-shaped	star-shaped
1 146	687	352	432

Table 5: Number of PFED$_{min}$ generated using Authorities and Classes summaries

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

Does PFED *generate 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. 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 6).

We generate star-shaped plausible federated queries based on *skos:prefLable* from SWDF and *dbpedia:thumbnail* from DBPedia (see table 6). 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 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 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)