UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Gabriela MONTOYA

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 11 mars 2016

Answering SPARQL Queries using Views

JURY

Rapporteurs :	**M. Bernd AMANN**, Professeur, Université de Pierre et Marie Curie (Paris 6) **M. Fabien GANDON**, Directeur de Recherche, INRIA Sophia Antipolis
Examinateurs :	**M. Philippe LAMARRE**, Professeur, INSA Lyon **M^me^ Pascale KUNTZ**, Professeur, Université de Nantes **M^me^ Maria-Esther VIDAL**, Professeur, Universidad Simón Bolívar
Directeur de thèse :	**M. Pascal MOLLI**, Professeur, Université de Nantes
Co-encadrante :	**M^me^ Hala SKAF-MOLLI**, Maître de Conferences, Université de Nantes

Acknowledgements

This PhD thesis was funded by the CNRS (Unit UMR6241).

Experiments presented in Chapter 8 were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

1

Introduction

1.1 Context

The Semantic Web is an extension of the Web, where information has precise meaning, and machines are able to understand the information and perform sophisticated tasks for the users [12]. In order to achieve the potential of the Semantic Web, the World Wide Web Consortium (W3C) has defined standards [75]: (i) for the representation of information on the Semantic Web, the Resource Description Framework (RDF) [52]; (ii) for querying the Semantic Web, the SPARQL language [66]; and (iii) for defining richer representations that contemplate intrinsic aspects of the reality, it has defined the Web Ontology Language (OWL) [63]. The number of sources has greatly increased in the last years, e.g., from 2011 to 2014 the increase was of 271% [72], and different actors of the society have published semantic data: scientific publications, e.g. the DBLP Bibliography Database [1], media, e.g., the Jamendo music repository [2], geography, e.g. the Norwegian geo-divisions [3], government, e.g., the UK transport dataset [4], life science, e.g., the UniProt dataset [5], and social networking, e.g., foaf

1. http://dblp.l3s.de/d2r/, November, 2015.
2. http://dbtune.org/jamendo/sparql/, November, 2015.
3. http://data.lenka.no/sparql, November, 2015.
4. http://openuplabs.tso.co.uk/sparql/gov-transport, November, 2015.
5. http://sparql.uniprot.org/, November, 2015.

profiles [6].

The Linked Data is a set of best practices for publishing and connecting structured data on the Web [13]. More than one thousand sources with semantic data have been crawled. [7] For many of these sources, there are infrastructures to execute SPARQL queries called *SPARQL endpoints*. [8] These endpoints allow users to explore datasets, and existing federated SPARQL query engines, such as FedX [74], ANAPSID [2] and SPLENDID [32], can use these endpoints to process SPARQL queries that require data from several endpoints to produce answers, i.e., *federated queries*, without having to move the data.

Some exemplar Semantic Web applications are the Linked Data Search Engines such as Swoogle [25], Sindice [61], and Watson [23]. These engines allow applications to traverse the Semantic Web, and retrieve meaningful and relevant data. Other exemplar Semantic Web application is service matchmaking [51], where semantic annotations are used in an e-commerce scenario where seekers search advertisers that satisfy a given set of characteristics. Another exemplar Semantic Web application is link prediction [78], where existing links among entities are used to propose new potential links, and focus the expert efforts on testing these potential links. Some other examples of Semantic Web applications are the analysis of governmental reforms [16], and health monitoring in smart houses [67].

The advantages that the Semantic Web and Linked Data have, with respect to Databases and the traditional Web, are the openness and meaningfulness of data. Openness because, as in the Web for documents, any piece of data can be linked to existing data, and links among data can be followed to discover new data. Meaningfulness because, as in Databases for entities, if two pieces of data are linked, their link has a precise meaning, and several types of links are possible to appropriately represent different kinds of relations.

We are interested by two issues in the context of Semantic Web. First, even if there is a large number of Linked Data sources, many sources from the Web [37] cannot be queried in conjunction with Linked Data sources using SPARQL, and this significantly reduces the space of queries that can be answered. Second, the low data availability provided by SPARQL endpoints [81] that prevents Semantic Web applications from relying on these infrastructures.

To address our first issue, **integration of Deep Web sources with Linked Data to answer**

6. e.g., http://www.w3.org/People/Berners-Lee/card.rdf, November, 2015.
7. According to the 2014 report about the state of Linking Data Cloud available at http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/
8. 615 SPARQL endpoints are registered at http://datahub.io/ (July, 2015)

SPARQL queries, there are two approaches. In the Data Warehousing approach [77], sources available in the Web can be transformed into RDF data using platforms such as Datalift [71], and their data can be linked with Linked Data sources using frameworks such as Silk [84]. However, this strategy to transform the Web sources into RDF has some limitations: (i) this transformation and linking should be done for all the known sources before executing any query; (ii) each time a source changes, transformation and linking of the whole source needs to be repeated to avoid stale answers. In the Mediators and Wrappers approach [85], mediators can be used to integrate data without having to move it from sources to the clients, therefore up to date data is queried. Among the mediator paradigms, the best suited for dynamic contexts, such as the Web, is the Local-as-View (LAV) paradigm [1]. Nevertheless, LAV traditional techniques used to produce query answers, *query rewritings*, may be too expensive in the context of SPARQL queries and numerous sources [56]. We are interested by strategies that are able to produce query answers against a set of LAV views, but without using query rewritings, i.e., strategies that load the data available through these views into an RDF graph, and execute the query against this RDF graph. These strategies do move data from sources to the mediator, but only the data from the selected views and only during query execution. In this thesis, we address this issue, and in particular the following research question:

Research Question 1. *In which order should the query relevant views be loaded into a graph, built during query execution, in order to use this graph to answer the query, and outperform the traditional LAV query rewriting techniques in terms of number of answers produced by time unit?*

To answer this research question, we propose the SemLAV approach. SemLAV integrates heterogeneous data from Linked Data and Deep Web, following the Local-as-View (LAV) paradigm [1] to describe the setup. Relevant views are ranked according to their possible contribution to the answers, and they are loaded into a graph instance, built during query execution, to answer SPARQL queries.

We have two contributions that address this first issue. Our first contribution is the formalization of the Maximal Coverage problem (MaxCov), it consists in selecting the k views to load in order to cover the greater number of rewritings. Our second contribution is the SemLAV relevant view selection and ranking algorithm, this algorithm sorts views according to the number of query subgoals that they cover, and ranks first the views that cover more query subgoals. It allows to load sources that may contribute more to the query answer first, and consequently to produce answers as soon as possible. Experimental results suggest that SemLAV outperforms traditional query rewriting

strategies. These contributions have been published in [56].

To address our second issue, **the SPARQL endpoints poor availability**, several strategies may be used. The Linked Data Fragments (LDF) [81] have been proposed to exploit client resources to relieve server resources and improve data availability. However, as each client has to perform most of the query processing, this strategy decreases the query performance in terms of number of answers produced by time unit, and amount of transferred data from servers to clients. A distributed query processing strategy to improve data availability is to give data consumers a more active role, and use their resources to replicate data, and consequently increase the data availability [45]. However, as Linked Data consumers are autonomous participants, the replication cannot closely follow the techniques used in distributed query processing, and new strategies to select and localize sources are needed. In this thesis, we are interested in improving data availability by using replication, and in particular the use of fragment replication. Replicating fragments leads to concerns about performance of query processing. For instance, if replicated fragments from popular sources are available through many endpoints, how are these endpoints going to be used to execute a federated query? A very simple solution may be to declare all these endpoints as part of the federation used by the federated query engine. However, this simple solution may incur in high execution time because redundant data would be transferred from endpoints to the federated query engine, and the federated query engine would have to execute the query joins. For example, executing a DBpedia query against a federation with one or two copies of DBpedia, leads to an increase of two orders of magnitude in the execution time for federated query engines FedX [74] and ANAPSID [2] as shown in Section 8.1. In order to properly exploit the benefits of replicated fragments, we propose a source selection strategy that is aware of fragment replication, and is able to enhance federated query processing engines. This idea of exposing replicated fragments through Linked Data endpoints is new, and so there are no existing techniques that are able to perform a source selection aware of data replication. Even if techniques to detect data overlapping based on data summaries exist [38, 70], applying them to scenarios with data replication will incur in expensive computations that are unnecessary in a replication scenario.

In this thesis, we address this issue, and want to answer the following research questions:

Research Question 2. *Can the knowledge about fragment replication be used to reduce the number of selected sources by federated query engines while producing the same answers?*

Research Question 3. *Does considering groups of triple patterns to be executed together, instead*

of individual triple patterns, produce source selections that lead to transfer less data from endpoints to the federated query engine?

To answer these research questions, we propose the FEDRA approach. FEDRA selects the sources to be contacted to evaluate each triple pattern in order to produce the query answers, this selection aims to improve the query performance in terms of the number of transferred tuples during query execution.

We have two contributions that address this second issue. Our first contribution is the formalization of the Source Selection Problem with Fragment Replication (SSP-FR). Given a federation of SPARQL endpoints that have replicated fragments, and a SPARQL query, it consists in selecting the sources that have to be contacted to retrieve data for each query triple pattern, such that the number of transferred tuples from sources to the federated query engine is reduced, and all the answers obtainable using the federation data are produced. Our second contribution is the FEDRA source selection algorithm, this algorithm approximates the SSP-FR problem. It uses query containment and equivalence among the fragment definitions to prune sources that provide redundant data, and an heuristic for set covering [41] to reduce the number of different endpoints used to retrieve data for the triple patterns of a basic graph pattern. State-of-the-art federated query engines, ANAPSID [2] and FedX [74], have been extended with FEDRA source selection strategy, and empirical results show that FEDRA enhances the federated query engines, and mostly reduces the number of selected sources and number of transferred tuples. These contributions have been published in [59].

1.2 Outline

This thesis is composed of two parts. Part I presents our contributions to address the issue of **integration of Deep Web sources with Linked Data to answer SPARQL queries**, while Part II presents our contributions to address the issue of **the SPARQL endpoints poor availability**. Chapter 2 presents background concepts related to the Semantic Web, and conjunctive queries, that are used through the thesis. Readers familiar with the Semantic Web technologies and conjunctive queries terminology may want to skip Chapter 2. In addition to this background chapter, each part has its own background sections: Section 5.1 for Part I and Section 8.1 for Part II.

1.2.1 Part I: Answering SPARQL queries using Linked Data and Deep Web sources

— Chapter 3 gives an introduction to the *Answering SPARQL queries using Linked Data and Deep Web sources* part.

— Chapter 4 presents state of the art for querying the Web of Data, Data Integration, and Query Rewriting.

— Chapter 5 defines the MaxCov problem, SemLAV query execution approach, algorithms, and experimental results.

1.2.2 Part II: Answering SPARQL Queries against Federations with Replicated Fragments

— Chapter 6 gives an introduction to the *Answering SPARQL Queries against Federations with Replicated Fragments* part.

— Chapter 7 presents state of the art for Distributed Databases and Linked Data query processing, source selection strategies for federated queries, and strategies to overcome availability limitations in Linked Data.

— Chapter 8 defines the SSP-FR problem, the FEDRA source selection algorithm, and some experimental results.

1.3 Publications list

This work led to the following publications:

1. Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Federated SPARQL Queries Processing with Replicated Fragments. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference*, pages 36–51, Bethlehem, United States, October 2015

2. Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. SemLAV: Local-As-View Mediation for SPARQL Queries. *Transactions on Large-Scale Data- and Knowledge-Centered Systems XIII*, pages 33–58, 2014

3. Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. SemLAV: Querying Deep Web and Linked Open Data with SPARQL. In *ESWC: Extended Semantic Web Conference*, volume 476 of *The Semantic Web: ESWC 2014 Satellite Events*, pages 332 – 337, Anissaras/Hersonissou, Greece, May 2014. This work is a demonstration of the work in [56].

4. Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Parallel data loading during querying deep web and linked open data with SPARQL. In Thorsten Liebig and Achille Fokoue, editors, *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, PA, USA, October 11, 2015.*, volume 1457 of *CEUR Workshop Proceedings*, pages 63–74. CEUR-WS.org, 2015. This work proposes an optimization of [56].

I also collaborated in two other papers that are not detailed in this dissertation:

— Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Gun: An efficient execution strategy for querying the web of data. In Hendrik Decker, Lenka Lhotská, Sebastian Link, Josef Basl, and A Min Tjoa, editors, *DEXA (1)*, volume 8055 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2013. This work leaded to the work in [56].

— Maria-Esther Vidal, Simon Castillo, Maribel Acosta, Gabriela Montoya, and Guillermo Palma. On the Selection of SPARQL Endpoints to Efficiently Execute Federated SPARQL Queries. *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, 2015. This paper extends the source selection work done before starting this PhD thesis [60].

And presented this thesis work at the doctoral consortium paper:

— Gabriela Montoya. Answering SPARQL Queries using Views. *ISWC-DC 2015 The ISWC 2015 Doctoral Consortium*, pages 33–40, 2015.

2

Background

In this chapter we present background concepts related to the Semantic Web and conjunctive queries. In the following section, standards used to represent data in the Semantic Web, and query these data are presented.

2.1 Semantic Web

The World Wide Web Consortium (W3C) [1] has developed standards to allow machines to understand the semantics behind the data published on the Web, increasing their possible interactions, and enhancing their data processing capabilities. Data enhanced with semantics, *Linked Data*, rely on Semantic Web technologies such as RDF, SPARQL, OWL and SKOS. Data is stored using the common format RDF, queries are posed using the standard language SPARQL, and vocabularies can be built using ontologies (e.g., using OWL). The Resource Description Framework (RDF) [52] is a framework for representing information in the Web. The basic unit of information are RDF triples, henceforth called *triples*. Each triple consists of a subject, a predicate and an object. The predicate describes the subject with a given characteristic whose value is given by the object. For example, in Listing 2.1, a film is described with two characteristics, first, its director, and second its name. Each resource

1. http://www.w3.org, July 2015.

is identified using IRIs, e.g., the director is identified by IRI http://dbpedia.org/resource/The_Rules_of_the_Game, also the predicates or properties are identified by IRIs, e.g., the film director is identified by the IRI http://dbpedia.org/ontology/director. Objects may be IRIs, e.g., http://dbpedia.org/resource/Jean_Renoir, or literals, e.g., "The Rules of the Game"@en. Literals may be strings, numbers, dates, etc. Strings may be annotated using language tags, e.g., '@en' indicates that the string language is English.

Listing 2.1 – Two RDF triples that describe a film

```
<http://dbpedia.org/resource/The_Rules_of_the_Game>
        <http://dbpedia.org/ontology/director> <http://dbpedia.org/resource/Jean_Renoir> .
<http://dbpedia.org/resource/The_Rules_of_the_Game>
        <http://dbpedia.org/property/name> "The Rules of the Game"@en .
```

IRIs in Listing 2.1 are part of DBpedia vocabularies [2]. DBpedia resources have IRIs that start with "http://dbpedia.org/resource/", this common prefix is called namespace IRI, and it may be associated with a namespace prefix, for "http://dbpedia.org/resource/" we have the namespace prefix "dbr", and the resource <http://dbpedia.org/resource/Jean_Renoir> may be written as dbr:Jean_Renoir using the namespace prefix. Listing 2.2 shows the same triples of Listing 2.1 using namespace prefixes.

Listing 2.2 – Two RDF triples that describe a film, with prefixes

```
PREFIX dbr:<http://dbpedia.org/resource/>
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX dbp:<http://dbpedia.org/property/>
dbr:The_Rules_of_the_Game dbo:director dbr:Jean_Renoir .
dbr:The_Rules_of_the_Game dbp:name "The Rules of the Game"@en .
```

When the prefixes are well-known or clear from the context, their declaration may be omitted. A RDF graph is defined as a set of RDF triples. And a RDF dataset is composed by a set of RDF graphs.

Once data is represented using the RDF framework, it may be queried using the SPARQL language. For instance, if we want to know who is the director of "The Rules of the Game" film, we may obtain it using the query in Listing 2.3. This query is composed of two triple patterns. Each triple pattern is composed of a subject, a predicate and an object as an RDF triple is, but each of these components may be a variable. For example, the first triple pattern has as subject the variable ?film, and as object the variable ?director. Variables are denoted by strings that start with character

2. http://wiki.dbpedia.org/, October 2015.

'?' or '$', in this thesis we will use variables starting with character '?'. The two triple patterns in Listing 2.3 compose a basic graph pattern. A basic graph pattern is a group of triple patterns, with no other operator between than '.', this operator represents the conjunction, i.e., both triple patterns should be satisfied, and common variables among the triples represent a join condition of equality.

Listing 2.3 – SPARQL SELECT query that retrieves the director of a film

```
SELECT ?director WHERE {
  ?film dbo:director ?director .
  ?film dbp:name "The Rules of the Game"@en
}
```

The answer to a SELECT query, as the one given in Listing 2.3, is a set of mappings. A mapping is a pair (variable, value). This set of mappings indicates the values that the variables in the WHERE should be instantiated to in order to obtain triples that belong to queried dataset. For example, the query in Listing 2.3, evaluated against the dataset in Listing 2.1, has as answer { (director, http://dbpedia.org/resource/Jean_Renoir) }.

Listing 2.4 – SPARQL ASK query that checks if there are triples with the director of a given film

```
ASK {
  ?film dbo:director ?director .
  ?film dbp:name "The Rules of the Game"@en
}
```

Besides SELECT queries, there are three other query types in SPARQL: ASK, CONSTRUCT and DESCRIBE queries. ASK queries have a query pattern, like the one included in the WHERE clause of SELECT queries, and its answer is a boolean value. Its answer is `true` if and only if there is a set of mappings such that the query triple patterns with their variables instantiated to the values in the set of mappings correspond to triples that belong to the queried dataset. For example, the query given in Listing 2.4, evaluated against the dataset in Listing 2.1, has as answer `true`.

Listing 2.5 – SPARQL CONSTRUCT query that returns a graph with the directed films

```
CONSTRUCT {
  ?director <http://example.org/directs> ?film
} WHERE {
  ?film dbo:director ?director .
}
```

Listing 2.6 – Answer to the SPARQL query in Listing 2.5, evaluated against the dataset in Listing 2.1

```
@prefix ns0:    <http://example.org/> .
@prefix dbr:    <http://dbpedia.org/resource/> .
dbr:Jean_Renoir ns0:directs     dbr:The_Rules_of_the_Game .
```

Listing 2.7 – SPARQL CONSTRUCT query that returns a graph with the film directors

```
CONSTRUCT {
  ?film dbo:director ?director
} WHERE {
  ?film dbo:director ?director
}
```

Listing 2.8 – SPARQL CONSTRUCT query that returns a graph with the film directors, using abbreviation for CONSTRUCT queries with graph template and pattern that are equal

```
CONSTRUCT WHERE {
  ?film dbo:director ?director
}
```

Listing 2.9 – Answer to the SPARQL queries in Listings 2.7 and 2.8, evaluated against the dataset in Listing 2.1

```
@prefix dbo:<http://dbpedia.org/ontology/> .
@prefix dbr:    <http://dbpedia.org/resource/> .
dbr:The_Rules_of_the_Game dbo:director dbr:Jean_Renoir.
```

CONSTRUCT queries have a graph template and a graph pattern, the graph pattern as in SELECT queries is introduced after the WHERE keyword. CONSTRUCT queries return a RDF graph. The variables instantiations obtained from the graph pattern are used to instantiate the variables in the graph template, and the ground triples, i.e., triples with only IRIs or literals, obtained from the graph template are combined in a single RDF graph. Listing 2.5 presents an example of a CONSTRUCT query with graph template and pattern that are different, its answer is presented in Listing 2.6. Notice that the variables used in the graph template are also used in the graph pattern. Listing 2.7 shows an example where the graph template and pattern are equal, in such cases the query may be abbreviated as in Listing 2.8, the answer to this query, evaluated against the dataset in Listing 2.1, is presented in Listing 2.9.

Listing 2.10 – SPARQL DESCRIBE query that returns a graph with a film description

```
DESCRIBE <http://dbpedia.org/resource/The_Rules_of_the_Game>
```

Listing 2.11 – SPARQL DESCRIBE query that returns a graph with films and directors description

```
DESCRIBE ?film ?director
WHERE {
  ?film dbo:director ?director
}
```

Listing 2.12 – Answer to the SPARQL queries in Listings 2.10 and 2.11, evaluated against the dataset in Listing 2.1

```
@prefix dbo:<http://dbpedia.org/ontology/> .
@prefix dbp:<http://dbpedia.org/property/> .
@prefix dbr:    <http://dbpedia.org/resource/> .
dbr:The_Rules_of_the_Game dbo:director dbr:Jean_Renoir.
dbr:The_Rules_of_the_Game dbp:name "The Rules of the Game"@en .
```

Finally, DESCRIBE queries return a single RDF graph that contains triples that describe one or more resources. The exact triples to be used for the description depend on the SPARQL query processor used. Some examples of DESCRIBE queries are given in Listings 2.10 and 2.11, they have the same answer when evaluated against the dataset in Listing 2.1, and their answer is presented in Listing 2.12.

Besides joins, other operators may be present in SPARQL queries, e.g., UNION or OPTIONAL. Comprehensive descriptions of the SPARQL language are presented in [66] and [64].

For the first issue that we address in this thesis, the integration of Deep Web sources with Linked Data to answer SPARQL queries, many of the approaches and algorithms presented in the Chapter 4 have been proposed for Conjunctive Queries. In the next section, basic notions about Conjunctive Queries are introduced in order to provide a common terminology to be used in Chapters 4 and 5.

2.2 Conjunctive Queries

A *conjunctive query* has the form: $Q(\bar{X})$:- $p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})$, where p_i is a predicate, $\bar{X_i}$ is a list of variables and constants, $\bar{X}$ is a list of variables, Q is the query name, $Q(\bar{X})$ is the *head* of the query, $p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})$ is the *body* of the query, and each element of the body, $p_i(\bar{X_i})$, is a *query subgoal*. In a conjunctive query, *distinguished variables* are variables that appear in the head. A conjunctive query is safe if all the distinguished variables also appear in the body of the query. Variables that appear in the body, but not in the head are *existential variables*. Variables are denoted by strings that start with a uppercase letter as $X1$ and constants by strings starting with

a lowercase letter as $a1$. An answer to a conjunctive query is a tuple $< v_1, ..., v_m >$ with a value for each $X_i \in \bar{X}$ for $1 \leq \text{i} \leq \text{m}$, such that every $p_j(\bar{X}_j)$, $1 \leq \text{j} \leq \text{n}$, evaluates to `true` when X_i is replaced by v_i.

Listing 2.13 – Conjunctive Query

```
q(X, Y) :- arc(X, Z), arc(Z, Y)
```

Listing 2.14 – Database

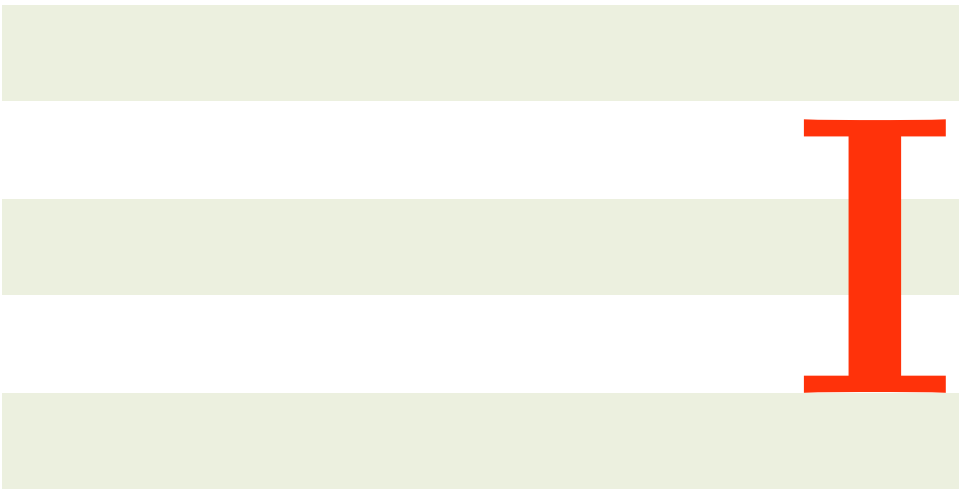
```
arc(a1, a2)
arc(a2, a3)
arc(a1, a4)
```

Listing 2.13 shows a conjunctive query that *finds the paths of size two*, and when it is executed against database in Listing 2.14, it produces *<a1, a3>*. Because *arc(a1, a2)* and *arc(a2, a3)* hold, then *<a1, a3>* is an answer to q, where the value *a1* is associated to the variable X, and the value *a3* is associated to the variable Y.

2.3 Summary

Standards have been developed by the Semantic Web community, in order to allow applications, to understand and perform tasks over the semantic data available on the Web. The standard for representing data, RDF, represents data using triples, i.e., *subject predicate object*. The predicate describes the subject with a given characteristic whose value is given by the object. The standard for querying data, the SPARQL language, is based on graph matching, and its basic components are the triple patterns. The triple patterns, differently from triples, allow the use of variables as subject, predicate or object. The evaluation of a graph pattern consists in finding values for the variables in the graph pattern, such that the resulting triples, after replacing variables by values, are actual triples in the queried set of RDF triples.

A conjunctive query is composed of a head and a body. The head has the form $q(X1, ..., Xn)$, with q the query name, and $X1, ..., Xn$ a list of variables. The body is composed of query subgoals, each subgoal is composed of a predicate and a list of arguments, and each argument may be a variable or a constant. The evaluation of a conjunctive query is a set of tuples $< v1, ..., vn >$, such that $v1, .., vn$ are constants that make the query subgoals, when variables $X1, ..., Xn$ are replaced by these constants, facts in the database.



Answering SPARQL queries using Linked data and Deep Web sources

3

Introduction

Processing queries over a set of autonomous and semantically heterogeneous data sources is a challenging problem. Particularly, a great effort has been made by the Semantic Web community to integrate datasets into the Linking Open Data (LOD) cloud [13] and make these data accessible through SPARQL endpoints which can be queried by federated query engines. However, there are still a large number of data sources and Web APIs that are not part of the LOD cloud. As consequence, existing federated query engines cannot be used to integrate these data sources and Web APIs. Supporting SPARQL query processing over these environments would extend federated query engines into the Deep Web.

Two main approaches exist for data integration: data warehousing and mediators. In data warehousing, data are transformed and loaded into a repository; this approach may suffer from the freshness problem [1], i.e., loaded data may produce stale answers to the queries if the source data have been updated. In the mediator approach, there is a global schema over which the queries are posed, and views that describe the relation between the global and source schema. Three main paradigms are proposed: Global-As-View (GAV), Local-As-View (LAV) and Global-Local-As-View (GLAV). In GAV mediators, relations of the global schema are described using views over the sources' schema, and including or updating sources may require the modification of a large number of views [79].

Whereas, in LAV mediators, the sources are described as views over the global schema, and adding new data sources can be easily done [79]. Finally, GLAV is a hybrid approach that combines both LAV and GAV approaches. GAV is appropriate for query processing in stable environments. A LAV mediator relies on a query rewriter to translate a mediator query into the union of conjunctive queries against the views. Therefore, it is more suitable for environments where data sources frequently change. Despite of its expressiveness and flexibility, LAV suffers from well-known drawbacks: (i) existing LAV query rewriters only manage conjunctive queries, (ii) the query rewriting problem is NP-complete for conjunctive queries, and (iii) the number of conjunctive queries that compose the query rewriting may be exponential.

SPARQL queries exacerbate LAV limitations, even in presence of conjunctions of triple patterns. For example, in a traditional database system, a LAV mediator with 140 conjunctive views can generate rewritings composed of 10,000 conjunctive queries for a conjunctive queries with eight subgoals [44]. In contrast, the number of queries that compose rewritings for a SPARQL query can be much larger. SPARQL queries are commonly comprised of a large number of triple patterns and some may be bound to *general predicates* of the RDFS or OWL vocabularies, e.g., `rdf:type`, `owl:sameAs` or `rdfs:label`, which are usually used in the majority of the data sources. Additionally, queries can be comprised of several star-shaped sub-queries [83]. Finally, a large number of variables can be projected out. All these properties emphasize the exponential complexity of the query rewriting problem, even enumerating the conjunctive queries in the rewritings can be unfeasible. For example, a SPARQL query with 12 triple patterns that comprises three star-shaped sub-queries can be rewritten using 476 views in rewritings that composed of billions of conjunctive queries. This problem is even more challenging considering that statistics may be unavailable, and there are no clear criteria to rank or prune the queries that compose the generated rewritings [74]. It is important to note that for conjunctive queries, GLAV query processing tasks are at least as complex as LAV tasks [20].

In this work, we focus on the LAV approach, and propose SemLAV, the first scalable LAV-based approach for SPARQL query processing. Given a SPARQL query Q on a set M of LAV views, SemLAV selects relevant views for Q and ranks them in order to maximize query results. Next, data collected from selected views are included into a partial instance of the global schema, where Q can be executed whenever new data is included; and thus, SemLAV incrementally produces query answers. Compared to a traditional LAV approach, SemLAV avoids generating rewritings

which is the main cause of the combinatorial explosion in traditional rewriting-based approaches; SemLAV also supports the execution of SPARQL queries. The performance of SemLAV is no more dependent on the number of conjunctive queries that compose the rewritings, but it does depend on the number and size of relevant views. Space required to temporarily include relevant views in the global schema instance may be considerably larger than the space required to execute all the queries that compose the query rewriting one by one. Nevertheless, executing the query once on the partial instance of the global schema could produce the answers obtained by executing all the queries that compose the query rewriting. Overall SemLAV should provide better performance than traditional LAV approaches in terms of number of answers produced by time unit (throughput), and time of the first answer. Moreover, SemLAV is capable of answering queries with UNIONs or OPTIONALs while traditional LAV approaches are not. Furthermore, SemLAV performance will be negatively impacted in terms of memory usage only if the selected views are not selective, as in that case, it risks to fill up the available memory.

To empirically evaluate the properties of SemLAV, we conducted an experimental study using the Berlin Benchmark [14] and queries and views designed by Castillo-Espinola [21]. Results suggest that SemLAV outperforms traditional LAV-based approaches with respect to answers produced per time unit, and provides a scalable LAV-based solution to the problem of executing SPARQL queries over heterogeneous and autonomous data sources.

The contributions of this part are the following:

— Formalization of the problem of finding the set of relevant LAV views that maximize query results; we call this problem MaxCov.

— A solution to the MaxCov problem.

— A scalable and effective LAV-based query processing engine to execute SPARQL queries, and to produce answers incrementally.

3.1 Outline of this part

Chapter 4 presents state of the art for querying the Web of Data, Data Integration, and Query Rewriting. Chapter 5 defines the MaxCov problem, SemLAV query execution approach, algorithms, and experimental results. Experimental results suggest that SemLAV outperform traditional query

rewriting strategies.

4

State of the Art

4.1 Querying the Web of Data

In recent years, several approaches have been proposed for querying the Web of Data [2, 11, 35, 36, 46]. Some tools address the problem of choosing the sources that can be used to execute a query [36, 46]; others have developed techniques to adapt query processing to source availability [2, 36]. Finally, frameworks to retrieve and manage Linked Data have been defined [11, 36], as well as strategies for decomposing SPARQL queries against federations of endpoints [74]. All these approaches assume that queries are expressed in terms of RDF vocabularies used to describe the data in the RDF sources; thus, their main challenge is to effectively select the sources from a catalog of known sources (or discover and select in the case of [36, 46]), and efficiently execute the queries on the data retrieved from the selected sources.

4.2 Data Integration

A data integration system, can be defined in terms of the mediated schema, henceforth global schema, the sources, and the mappings between sources and the global schema.

Definition 1 (Data Integration System [19]). *A data integration system $\mathcal{I}$ is a triple $<\mathcal{G}, \mathcal{S}, \mathcal{M}>$ where*

— *$\mathcal{G}$ is the global schema, expressed in the relational model, possibly with constraints*

— *$\mathcal{S}$ is the source schema, also expressed in the relational model.*

— *$\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of assertions of the form $q_{\mathcal{S}} \subseteq q_{\mathcal{G}}$, where $q_{\mathcal{S}}$, $q_{\mathcal{G}}$ are two queries of the same arity, over the source schema $\mathcal{S}$ and global schema $\mathcal{G}$ respectively.*

Querying heterogeneous data sources has been performed in Databases using two main approaches: Data Warehousing [77], and Mediators and Wrappers [85].

4.2.1 Data Warehousing

In Data Warehousing the data is retrieved from the sources, transformed into the warehouse schema, and stored in a repository before executing any query. In this context, query optimization relies on materialized views that allow to speed up the execution time. Selecting the best set of views to be materialized is a complex problem that has been deeply studied in the literature [21, 33, 22, 42, 30]. Commonly approaches attempt to select this set of views according to an expected workload and available resources. These approaches exhibit good performance for the queries that can be rewritten using the materialized views, but not necessarily for the other queries. Further, the cost of the view maintainability process can be very high if the data frequently change, and it needs to be kept up-to-date to ensure answer correctness.

4.2.2 Mediators and Wrappers

In Mediators and Wrappers, the data do not have to be moved from the sources. Queries are posed using the global schema, independently of how data is really stored in the sources. Wrappers transform data from the source schemas into instances of the global schema, and mediators produce the query answers.

Mappings between source and the global schemas are used to rewrite the user query into source queries. Some approaches have been proposed to write the descriptions and associated rewriting algorithms, the three main mediator paradigms that have been proposed to integrate dissimilar

data sources are: Global-As-View (GAV) [49], Local-As-View (LAV) [49], and Global-Local-As-View (GLAV) [29].

Sources may provide *sound*, *complete* or *exact* information with respect to the global schema instances obtainable from their data [48]. If their data correspond to a subset of the global schema, they are sound. If their data correspond to a superset, they are complete. And if their data correspond to both a subset and superset, they are exact. In the context of autonomous sources, like the Web, the sources are only assumed to be sound.

To illustrate the different approaches, consider an integration system that provides information about books. First, the system uses only two sources. Source *s1*, gives book titles and authors, while source *s2* provides reviewers and reviews from the read books.

The global schema is composed by the relations: book(title), author(title, author), hasRead(person, author), bookCategory(title, category), authorCategory(author, category), and reviews(document, review).

The query in Listing 4.1 asks for authors of books in the same category that they have read books.

Listing 4.1 – Conjunctive query that asks for authors who have written and read books of the same category

```
q(A1) :- hasRead(A1, A2), author(T1, A1), author(T2, A2), bookCategory(T1, C), bookCategory(T2, C)
```

GAV mediators

Definition 2 (Global As View (GAV) approach [49])**.** *In the GAV approach, for each relation R in the mediated schema, we write a query over the source relations specifying how to obtain R's tuples from the sources*

In terms of Definition 1, each element g in $\mathcal{G}$ is associated to a query $q_{\mathcal{S}}$ over $\mathcal{S}$ by one assertion in mapping $\mathcal{M}$, i.e., $q_{\mathcal{S}} \subseteq g$.

Listing 4.2 – GAV mappings when only *s1* and *s2* have been included in the system

```
book(T) ⊇ s1(T, A)
author(T, A) ⊇ s1(T, A)
hasRead(P, A) ⊇ s1(T, A), s2(P, R, T)
reviews(D, R) ⊇ s2(P, R, D)
```

In the example, the book, author, hasRead, and reviews relations may be written as views over the sources as in Listing 4.2. If a third source with the category of the books is added to the system, then it has to be considered that these new sources may have some common data with the sources that already belong to the system, and that new mappings, or modifications of existing mappings may have to be made. In the example, *s3* can be used with *s1* to make the new mappings given in Listing 4.3

Listing 4.3 – GAV mappings added after *s3* is included in the system

```
bookCategory(T, C) ⊇ s3(T, C)
authorCategory(A, C) ⊇ s1(T, A), s3(T, C)
```

Moreover, if a new source, *s4*, with book information is included, the mappings given in Listing 4.2 and 4.3 have to be modified, as shown in Listing 4.4. Multiple assertions for the same global schema element are included to provide alternative source queries, instead of only one assertion with a more complex query.

Listing 4.4 – GAV mappings after *s4* is included in the system

```
book(T) ⊇ s1(T, A)
book(T) ⊇ s4(T, A)
author(T, A) ⊇ s1(T, A)
author(T, A) ⊇ s4(T, A)
reviews(D, Review) ⊇ s2(P, Review, D)
hasRead(P, A) ⊇ s1(T, A), s2(P, Review, T)
hasRead(P, A) ⊇ s4(T, A), s2(P, Review, T)
bookCategory(T, C) ⊇ s3(T, C)
authorCategory(A, C) ⊇ s1(T, A), s3(T, C)
authorCategory(A, C) ⊇ s4(T, A), s3(T, C)
```

Including or updating data sources may require the modification of a large number of mappings [79], but answering query q is naturally done by unfolding the global schema relations present in the query, and substituting them by the source views.

Definition 3 (Query Unfolding [26]). *Given a query Q and a query subgoal $g_i(\bar{X}_i)$, $g_i(\bar{X}_i) \in body(Q)$, where g_i corresponds to a mapping: $g_i(\bar{Y})$:- $s_1(\bar{Y_1}), \ldots, s_n(\bar{Y_n})$, the unfolding of g_i in Q is done using a variable mapping τ from variables in $\bar{Y}$ to variables in $\bar{X}_i$, replacing $g_i(\bar{X}_i)$ by $s_1(\tau(\bar{Y_1})), \ldots, s_n(\tau(\bar{Y_n}))$ in Q. Variables that occur in the body of g_i but not in $\bar{X}_i$ are replaced by fresh (unused) variables by mapping τ.*

For instance, the query in Listing 4.1, q, can be rewritten, using the mappings in Listings 4.2 and 4.3, as shown in Listing 4.5. For query subgoal hasRead(A1,A2) and mapping hasRead(P, A) $\supseteq$ s1(T, A), s2(P, R, T), the variable mapping τ is defined as: $\tau(P) = A1$, $\tau(A) = A2$, $\tau(T) = T'$, $\tau(R) = R'$, with T' and R' two fresh variables.

Listing 4.5 – Conjunctive query q and its rewriting, r, in terms of the sources from Listings 4.2 and 4.3

```
q(A1) :- hasRead(A1, A2), author(T1, A1), author(T2, A2), bookCategory(T1, C), bookCategory(T2, C)
r(A1) :- s1(T', A2), s2(A1, R', T'), s1(T1, A1), s1(T2, A2), s3(T1, C), s3(T2, C)
```

LAV mediators

Definition 4 (Local As View (LAV) approach [49])**.** *In the LAV approach, the contents of a source are described as a query over the mediated schema relations.*

In terms of Definition 1, each element s in $\mathcal{S}$ is associated to a query $q_{\mathcal{G}}$ over $\mathcal{G}$ by one assertion in mapping $\mathcal{M}$, i.e., $s \subseteq q_{\mathcal{G}}$.

Listing 4.6 – LAV mappings when only *s1* and *s2* have been included in the system

```
s1(T, A) ⊆ book(T), author(T, A)
s2(P, Review, D) ⊆ book(D), hasRead(P, A), author(D, A), reviews(D, Review)
```

In the example, views *s1* and *s2* are defined in Listing 4.6. If *s3* and *s4* are added as before, only mappings involving *s3* or *s4* need to be added, and the existing mappings remain unchanged as shown in Listing 4.7.

Listing 4.7 – LAV mappings after *s3* and *s4* have been included in the system

```
s1(T, A) ⊆ book(T), author(T, A)
s2(P, Review, D) ⊆ book(D), hasRead(P, A), author(D, A), reviews(D, Review)
s3(T, C) ⊆ book(T), bookCategory(T, C), author(T, A), authorCategory(A, C)
s4(T, A) ⊆ book(T), author(T, A)
```

In the LAV approach, new data sources can be easily integrated [79]; further, data sources that publish entities of several concepts in the global schema, can be naturally defined as LAV views.

Answering q using the LAV mappings may be more complex, as they do not provide for each relation in q a source view to replace it. Instead of unfolding as it is the case for GAV mappings, query rewritings are used for LAV mappings. The following definitions about query containment and equivalence are used to formalize the notion of query rewriting.

Definition 5 (Query Containment and Equivalence [26] [34]). *Given two queries $Q1$ and $Q2$ with the same number of arguments in their heads, $Q1$ is contained in $Q2$, $Q1 \sqsubseteq Q2$, if for any database instance D the answer of $Q1$ over D is contained in the answer to $Q2$ over D, $Q1(D) \subseteq Q2(D)$. $Q1$ is equivalent to $Q2$ if $Q1 \sqsubseteq Q2$ and $Q2 \sqsubseteq Q1$.*

However, it is not practical to check the containment condition for any database instance D, and instead of that, the containment check is based on the existence of a containment mapping between the queries and a theorem that establishes the equivalence of containment and the existence of a containment mapping.

Definition 6 (Containment Mapping [26]). *Given two queries $Q1$ and $Q2$, $\bar{X}$ and $\bar{Y}$ the head variables of $Q1$ and $Q2$ respectively, and ψ a variable mapping from $Q1$ to $Q2$, ψ is a containment mapping if $\psi(\bar{X}) = \bar{Y}$ and for every query subgoal $g(\bar{X}_i)$ in the body of $Q1$, $\psi(g(\bar{X}_i))$ is a subgoal of $Q2$.*

Theorem 4.2.1 (Containment [26]). *Let $Q1$ and $Q2$ be two conjunctive queries, then there is a containment mapping from $Q1$ to $Q2$ if and only if $Q2 \sqsubseteq Q1$.*

Using the notions of containment and equivalence, the definitions of equivalent and maximally-contained rewritings are formalized. Notice that it is not always possible to find an equivalent rewriting, in particular given the assumption that sources are sound but not necessarily complete. Existing algorithms presented in Section 4.2.3 aim to find the maximally-contained rewriting.

Definition 7 (Equivalent Rewriting [34]). *Let Q be a query and $M = \{v_1, \ldots, v_m\}$ be a set of views definitions. The query Q' is an equivalent rewriting of Q using M if:*

— *Q' refers only to views in M, and*

— *Q' is equivalent to Q.*

Definition 8 (Maximally-Contained Rewriting [34]). *Let Q be a query, $M = \{v_1, \ldots, v_m\}$ be a set of views definitions, and L be a query language[1]. The query Q' is a maximally-contained rewriting of Q using M with respect to L if:*

— *Q' is a query in L that refers only to the views in M,*

— *Q' is contained in Q, and*

1. L is a query language defined over the alphabet composed of the global and source schema

— *there is no rewriting* $Q_1 \in L$*, such that* $Q' \sqsubseteq Q_1 \sqsubseteq Q$ *and* Q_1 *is not equivalent to* Q'*.*

Listing 4.8 presents the two queries, *r1* and *r2*, that rewrite q. The containment mapping ψ that can be used to show the containments $r1 \sqsubseteq q$ and $r2 \sqsubseteq q$ is defined as $\psi(A1) = A1$, $\psi(A2) = A'$, $\psi(T1) = T1$, $\psi(T2) = T2$, $\psi(C) = C$. *r1* and *r2* are contained rewritings of q, but they are not maximally-contained rewritings as they can be combined to produce the maximally contained rewriting of q: $\mathsf{r1} \bigcup \mathsf{r2}$.

Listing 4.8 – Conjunctive query q and its two contained rewritings in terms of the sources from Listing 4.7

```
q(A1) :- hasRead(A1, A2),  author(T1, A1), author(T2, A2), bookCategory(T1, C), bookCategory(T2, C)
r1(A1) :- s2(A1, R', T2), s1(T1, A1), s3(T1, C), s3(T2, C)
r2(A1) :- s2(A1, R', T2), s4(T1, A1), s3(T1, C), s3(T2, C)
```

GLAV mediators

Global-Local-As-View (GLAV), a generalization of LAV and GAV, has been proposed in [29]. GLAV allows the definition of mappings where views on the global schema are mapped to views of the data sources.

In terms of Definition 1, queries $q_{\mathcal{S}}$ over $\mathcal{S}$ are associated to queries $q_{\mathcal{G}}$ over $\mathcal{G}$ by assertions in mapping $\mathcal{M}$, i.e., $q_{\mathcal{S}} \subseteq q_{\mathcal{G}}$.

Listing 4.9 – Mappings in the Global Local As View approach

```
s1(T, A), s2(P, Review, T) ⊆ hasRead(P, A), reviews(T, Review)
s4(T, A), s2(P, Review, T) ⊆ hasRead(P, A), reviews(T, Review)
```

The mappings given in the previous sections are also GLAV mappings because they are GAV or LAV mappings. Moreover, mappings in Listing 4.9 are GLAV mappings, but they are neither LAV nor GAV mappings.

Recently, Knoblock et al. [43] and Taheriyan et al. [76] proposed Karma, a system to semi-automatically generate source descriptions as GLAV views on a given ontology. Karma makes GLAV views a solution to consume open data as well as to integrate and publish these sources into the LOD cloud. GLAV views are suitable not only to describe sources, but also to provide the basis for the dynamic integration of open data and Web APIs into the LOD cloud. Further, theoretical results presented by Calvanese et al. [20] establish that for conjunctive queries against relational schemas,

GLAV query processing techniques can be implemented as the combination of the resolution of the query processing tasks with respect to the LAV component of the GLAV views followed by query unfolding tasks on the GAV component.

4.2.3 LAV Query Rewriting Techniques

The problem of rewriting a query into queries over the data sources is a relevant problem in integration systems [50]. A great effort has been made to provide solutions able to produce query rewritings in the least time possible and to scale up to a large number of views. Several approaches have been defined, e.g., the Bucket algorithm [50], the MiniCon algorithm [65, 34], MCDSAT [10], and GQR [44].

The Bucket Algorithm

The Bucket algorithm [50, 34] is comprised of two parts. In the first part, for each query subgoal a bucket is created, and each bucket is filled with the views such that one of its view subgoals can cover the bucket query subgoal. Then, in the second part, the buckets are used to build contained rewritings. These rewritings are created by taking one view from each bucket. Then, their validity as rewriting is checked. A query is a valid rewriting if it is contained in the query, or may be contained in the query by adding predicates. A view subgoal sg_v is said to cover one query subgoal sg_q if the following conditions are satisfied:

— There is a variable mapping ψ such that $\psi(sg_v) = \psi(sg_q)$

— The mapping ψ applied to the variables in the view head makes the predicates appearing in the query and the view mutually satisfiable. If a query variable is in the position i of sg_q, and it is distinguishable, then if there is a variable in the position i of sg_v, it is also distinguishable.

Algorithm 1 presents the first part of the Bucket algorithm. In this algorithm, the functions *head(Q)* and *body(Q)* are used to retrieve the head and body of a conjunctive query Q; the predicate *distinguishable(var, Q)* is used to indicate that the variable *var* appears in *head(Q)*; the function *predicate(p(X))* returns the predicate p in the query subgoal $p(X)$; the function *argument(i, q)* returns the i-th argument in subgoal q; and the function *newVariable(Q, V)* returns a new variable name unused in the query Q or the set of views V.

Algorithm 1 CreateBuckets Algorithm [50, 34]

```
Require: V : set of View; m : integer; Q: ConjunctiveQuery (with m subgoals)
 1: procedure CREATEBUCKETS(V, Q)
 2:     for all i ∈ 1 ≤ i ≤ m do
 3:         Bucket_i ← ∅
 4:     end for
 5:     for all q ∈ body(Q) do
 6:         for all v ∈ V do
 7:             for all w ∈ body(v) do
 8:                 if predicate(q) = predicate(w) then
 9:                     if y = argument(k, w) ∧ distinguishable(y, v) then
10:                         ψ(y) = argument(k, q)
11:                     else
12:                         ψ(y) = newVariable(Q, V)
13:                     end if
14:                     if satisfiable(body(Q) ∧ (∀ p : p ∈ body(v) : ψ(p))) then
15:                         if (∀ a, i : distinguishable(a, Q) ∧ a = argument(i, q) : distinguishable(argument(i, w),v)) then
16:                             Bucket_i ← Bucket_i ⋃ { ψ(head(v)) }
17:                         end if
18:                     end if
19:                 end if
20:             end for
21:         end for
22:     end for
23: end procedure
```

First, a bucket for each query subgoal is initialized as empty (lines 2-4). Then, the views that may be used to cover a query subgoal are added in the query subgoal bucket (lines 5-22). For each query subgoal, each view subgoal is considered if they share the same predicate (line 8), and the mapping is built according to the condition of distinguishable of the variable in the view subgoal. If it is distinguishable, then the variable in the view subgoal is mapped to it, but if it is not, a new variable is mapped (lines 9-13). Then if the query subgoals and the view subgoals, with the variable replacement induced by the mapping, are mutually satisfiable (line 14),[2] and the distinguishable variables in the query are mapped to distinguishable variables in the view (line 15), then the view head with the variable replacement induced by the mapping is included in the bucket (line 16).

Proposition 1. *The time complexity of Algorithm 1 is $O(n \times m \times k \times l)$, where n is the number of query subgoals, m is the number of views, k is the maximum number of view goals, l is the maximum number of arguments per query or view subgoal*

Listing 4.10 – Conjunctive query that asks for authors that have read books from authors that write books in the same category than they do

```
q(A1) :- hasRead(A1, A2), author(T1, A1), author(T2, A2), bookCategory(T1,C), bookCategory(T2, C)
```

Listing 4.11 – LAV mappings

```
s1(T, A) ⊆ book(T), author(T, A)
```

2. For conjunctive queries as defined in Section 2.2 without inferences, these expressions are always mutually satisfiable, but the inclusion of constraints like Var > value may made them not mutually satisfiable.

```
s2(P, R, D) ⊆ book(D), hasRead(P, A), author(D, A), reviews(D, R)
s3(T, C) ⊆ book(T), bookCategory(T, C), author(T, A), authorCategory(A, C)
s4(T, A) ⊆ book(T), author(T, A)
```

For query given in Listing 4.10, and views in Listing 4.11, the algorithm builds five buckets, one for each query subgoal as in Table 4.1.

For the second subgoal, author(T1, A1), the view s1(T1, A1) has been included as the mapping $\psi(A) = A1$, $\psi(T) = T1$, makes *s1* second subgoal author(T, A), equal to the query subgoal, and distinguishable variable *A1* in the query corresponds to distinguishable variable A in the view. But view s3(T1, X7) has not been included in this bucket because the variable A, distinguishable in the query, can only correspond to an existential variable in the view.

Table 4.1 – Buckets for query in Listing 4.10, and views in Listing 4.11

hasRead(A1, A2)	author(T1, A1)	author(T2, A2)	bookCategory(T1, C)	bookCategory(T2, C)
s2(A1, X1, X2)	s1(T1, A1) s4(T1, A1)	s1(T2, A2) s2(X5, X6, T2) s3(T2, X8) s4(T2, A2)	s3(T1, C)	s3(T2, C)

In the second part of the Bucket algorithm [50, 34], the Cartesian product of the built buckets is considered. Each element of this Cartesian product is a possibly contained rewriting, having one element from each bucket to cover the corresponding query subgoal. Each possibly contained rewriting should satisfy two conditions to be a valid contained rewriting: (i) they should be satisfiable; (ii) they should be contained in the query. Join predicates can be added to the possible rewritings in order to make them contained in the query.

Listing 4.12 gives the eight possible contained rewritings. Rewriting *r1* includes the first element of each bucket to cover the query subgoals, rewriting *r2* includes the second element of the bucket for the third query subgoal, and the first element for all the other buckets. In *r2* the same view (*s2*) is used to cover the first and third subgoals, while the *r1* two different views, *s2* and *s1*, are used to cover the first and third subgoals.

Listing 4.12 – Cartesian product of the buckets in Table 4.1, these queries are the possibly contained query rewritings

```
r1(A1) :- s2(A1, X1, X2), s1(T1, A1), s1(T2, A2), s3(T1, C), s3(T2, C)
r2(A1) :- s2(A1, X1, X2), s1(T1, A1), s2(X5, X6, T2), s3(T1, C), s3(T2, C)
r3(A1) :- s2(A1, X1, X2), s1(T1, A1), s3(T2, X8), s3(T1, C), s3(T2, C)
r4(A1) :- s2(A1, X1, X2), s1(T1, A1), s4(T2, A2), s3(T1, C), s3(T2, C)
```

```
r5(A1) :- s2(A1, X1, X2), s4(T1, A1), s1(T2, A2), s3(T1, C), s3(T2, C)
r6(A1) :- s2(A1, X1, X2), s4(T1, A1), s2(X5, X6, T2), s3(T1, C), s3(T2, C)
r7(A1) :- s2(A1, X1, X2), s4(T1, A1), s3(T2, X8), s3(T1, C), s3(T2, C)
r8(A1) :- s2(A1, X1, X2), s4(T1, A1), s4(T2, A2), s3(T1, C), s3(T2, C)
```

All the possibly contained rewritings given in Listing 4.12 are satisfiable as there are no two predicates in the same query that can produce any contradiction. However, not all of them are contained in the query. The first query in Listing 4.12, *r1*, that uses view *s2* to cover the first subgoal and view *s1* to cover the third subgoal, is not contained in the query, *q*, given in Listing 4.10. The condition imposed on the first and third query subgoals with the shared variable *A2*, cannot be satisfied by view subgoals in *s2* and *s1*, because query variable *A2* has been mapped to a non distinguishable variable in *s2*. On the other hand, *r2*, that uses *s2* to cover the first and third query subgoals, can be contained in the query if the join predicates $X2 = T2$ and $A1 = X5$ are added to *s2*. Adding the constraints imposed by these join predicates can be also done by replacing variables *X2* and *X5* by *T2* and *A1*, furthermore, after replacing the variables one of the occurrences of view *s2* can be safely removed. Notice that the difference among the queries given in Listing 4.12, may be subtle, as it is the case for *r1* and *r2*. *s2* is already present in *r1* and it is only used to cover the first query subgoal, while in *r2* it is used to cover both the first and third query subgoals.

From the eight possible contained rewritings given in Listing 4.12, only the queries *r2* and *r6* are contained in *q*. Valid rewritings, after variable replacing and simplification, are given in Listing 4.13. The maximally contained rewriting is $\mathsf{r2} \bigcup \mathsf{r6}$.

Listing 4.13 – q's valid contained rewritings, r2 and r6, obtained from the queries given in Listing 4.12

```
q(A1) :- hasRead(A1, A2), author(T1, A1), author(T2, A2), bookCategory(T1,C), bookCategory(T2, C)
r2(A1) :- s2(A1, X1, T2), s1(T1, A1), s3(T1, C), s3(T2, C)
r6(A1) :- s2(A1, X1, T2), s4(T1, A1), s3(T1, C), s3(T2, C)
```

The MiniCon Algorithm

The MiniCon algorithm [65, 34] is an optimization of the Bucket algorithm that avoids the last verification step by a more complex first step. The MiniCon algorithm uses MiniCon Descriptors (MCDs) instead of buckets. The number of combinations of MCDs is considerably lower than for the buckets, and all the resulting rewritings are contained in the query by construction. For each query subgoal, if a view subgoal sg_v covers a query subgoal sg_q, all the query subgoals that share variables

with sg_q are considered together, and further checking is done to assess that these query subgoals may be covered by view subgoals in a compatible way. Then, a MCD is created and it includes the view head with the proper mapping, and the covered subgoals. To formally define MCDs, the term *head homomorphism* is used. A head homomorphish h for view V is a variable mapping from the variables in V to the variables in V, that is the identity on existential variables, but may make two distinguished variables equal.

Definition 9 (MiniCon descriptions [65]). *An MCD C for a query Q over a view V is a tuple of the form* $(h_C, V(\bar{Y})_C, \varphi_C, G_C)$ *where* h_C *is a head homomorphism on V,* $V(\bar{Y})_C$ *is the result of applying* h_C *to V, i.e.,* $\bar{Y}$=$h_C(\bar{A})$*, where* $\bar{A}$ *are the head variables of V,* φ_C *is a partial mapping from Vars(Q) to* h_C*(Vars(V)),* G_C *is a subset of the subgoals in Q which are covered by some subgoal in* h_C*(V) using the mapping* φ_C *(note: not all such subgoals are necessarily included in* G_C*).*

If G_C has the minimum size such that the conditions are satisfied, then a set of MCDs with disjoint subgoals can be built, and the combination of MCDs is straightforward. Query rewritings are obtained by combining MCDs such that all the query subgoals are covered. In order to reduce the number of MCDs combinations, the MiniCon algorithm obtains MCDs that satisfy Property 1.

Property 1 (Property 1 [65]). *Let C be an MCD for Q over V. Then C can only be used in a non-redundant rewriting of Q if the following conditions hold:*

C1 For each head variable x of Q which is in the domain of φ_C*,* $\varphi_C(x)$ *is a head variable in* h_C*(V).*

C2 If $\varphi_C(x)$ *is an existential variable in* h_C*(V), then for every g, subgoal of Q, that includes x: (1) all the variables in g are in the domain of* φ_C *; and (2)* $\varphi_C(g) \in h_C(V)$

When Property 1 is ensured, then rewriting construction is easily done thanks to Property 2.

Property 2 (Property 2 [65]). *Given a query Q, a set of views V, and the set of MCDs C for Q over the views in V, the only combinations of MCDs that can result in non-redundant rewritings of Q are of the form* C_1,...,C_l*, where:*

D1 $G_{C_1} \bigcup \ldots \bigcup G_{C_l} = Subgoals(Q)$*, and*

D2 for every $i \neq j$, $G_{C_i} \bigcap G_{C_j} = \emptyset$

Algorithm 2 presents the first part of the MCD algorithm. Similarly to the Bucket algorithm, for each query subgoal, each view and its goals are considered to cover the query subgoal (lines 3-12).

Algorithm 2 MiniCon first part: form MCDs [65]

Require: Q : ConjunctiveQuery; V: set of View (defined as ConjunctiveQuery)
Ensure: C: set of MCD
1: **function** FORMMCDs(Q, V)
2: $C \leftarrow \emptyset$
3: **for all** $q \in body(Q)$ **do**
4: **for all** $v \in V$ **do**
5: **for all** $w \in body(v)$ **do**
6: **if** There is a mapping φ and head homomorphism on V, h, such that $\varphi(q) = h(w)$ **then**
7: h $\leftarrow$ the least restrictive homomorphism h such that $\varphi(q) = h(w)$
8: C $\leftarrow$ C $\bigcup$ { $(h_C, V(\bar{Y})_C, \varphi_C, G_C)$: h $\subseteq h_C \wedge \varphi \subseteq \varphi_C \wedge (h_C, V(\bar{Y})_C, \varphi_C, G_C)$ is minimal for Property 1 }
9: **end if**
10: **end for**
11: **end for**
12: **end for**
13: **return** C
14: **end function**

Head homomorphism h and mapping φ are looked up (line 6), and their extensions that satisfying Property 1 cover the least number of query subgoals, are used to form the MCDs (line 8).

Proposition 2. *The time complexity of Algorithm 2 is $O(n^n \times m \times k^n \times l^n)$, where n is the number of query subgoals, m is the number of views, k is the maximum number of view goals, l is the maximum number of arguments per query or view subgoal.*

The first part of the MiniCon algorithm has higher time complexity than the first part of the Bucket algorithm, as the variables from the set of query subgoals that share existential variables should be mapped to a set of view subgoal variables to satisfy *C2* from Property 1. But the overall complexity for both algorithms is the same: $O((n \times m \times k)^n)$, where n is the number of query subgoals, m is the number of views, k is the maximum number of view goals, l is the maximum number of arguments per query or view subgoal (and l is dominated by k) [65].

Table 4.2 – MCDs for query in Listing 4.10 and views in Listing 4.11, for h and φ identity part has been omitted, i.e., h(X) = X (φ(X)=X) for any other variable in the domain of h (φ)

V($\bar{Y}$)	h	φ	G
s1(T, A)		T1→ T, A1 → A	2
s1(T, A)		T2→ T, A2 → A	3
s2(P, R, D)		A1→ P, A2 → A, T2→ D	1, 3
s3(T, C)		T1→ T	4
s3(T, C)		T2→ T	5
s4(T, A)		T1→ T, A1 → A	2
s4(T, A)		T2→ T, A2 → A	3

For the query given in Listing 4.10, and the views in Listing 4.11, the MiniCon algorithm builds four MCDs, as depicted in Table 4.2. For view *s4*, two MCDs have been built, one for the fourth subgoal and another for the fifth subgoal, as variable C in view *s4* is distinguishable and joins on

that variable can be enforced without having to cover both subgoals in the same MCD. Notice that no MDC has been built for view *s3* and the third subgoal. View *s3* is not included for the third subgoal because variable *A2* is existential in the view and the view does not cover all the query subgoals that involve the variable *A2*.

Listing 4.14 – Valid contained rewritings, r1 and r2, obtained from the combination of MCDs in Table 4.2

```
q(A1) :- hasRead(A1, A2), author(T1, A1), author(T2, A2), bookCategory(T1,C), bookCategory(T2, C)
r1(A1) :- s2(A1, X1, T2), s1(T1, A1), s3(T1, C), s3(T2, C)
r2(A1) :- s2(A1, X1, T2), s4(T1, A1), s3(T1, C), s3(T2, C)
```

Listing 4.14 presents the only two valid contained rewritings obtainable from the MCDs in Table 4.2, these rewritings are equivalent to the rewritings given in Listing 4.13, and obtained using the Bucket algorithm.

MCDSAT and SSD-SAT

MCDSAT [10] is a logic based method to produce MiniCon Descriptors (MCDs) and rewritings as translations of models for logical theories. These theories, called MCD theory and extended theory, model the rules that any MCD or rewriting must satisfy. These theories are compiled into d-DNNFs [24], i.e., deterministic, decomposable negation normal form, for which model counting can be done in polynomial time. This query rewriter benefits from existing d-DNNFs compilers to produce rewritings faster than the traditional MiniCon implementation [10].

Izquierdo et al [40] extend the MCDSAT rewriter with constants and preferences to identify the combination of semantic services that rewrite a user request. The expressive power of this extension is greater but also is the complexity of the logical theories.

Graph-based Query Rewriting (GQR)

Graph-based Query Rewriting (GQR) [44] models query and view subgoals as graphs. These graphs abstract from variable names, and a preprocessing step is performed over the views to compactly represent all the views with few graphs. Then, when a query is posed, for each query subgoal relevant graphs are selected, and graphs are incrementally combined in order to produce larger graphs that cover more query subgoals. The view graphs correspond to partial rewritings, and differently from previous rewriters, rewritings may be produced incrementally as the partial rewritings are ex-

tended to cover all the query subgoals. Additionally, GQR prunes the partial rewritings that cannot cover all the query joins in order to keep only the partial rewritings that can actually be extended to become valid rewritings. This is another advantage with respect to other rewriters, as bucket elements or MCDs may be produced even if they are not used in any valid rewriting.

4.2.4 GUN

GUN [55] is a strategy to maximize the number of answers obtained from a given set of k rewritings; GUN aggregates the data obtained from the relevant views present in those k rewritings and executes the query over it. Even if GUN can maximize the number of obtained answers, it still depends on query rewritings as input, and has no criteria to order the relevant views.

4.3 Summary

Answering SPARQL queries using RDF sources has been the object of several studies [2, 11, 35, 36, 46]. Data integration has been done in databases following two main approaches: data warehousing [77], and mediators and wrappers [85]. Data warehousing allows for the local optimization of queries, but data needs to be frequently updated to avoid stale answers, and it provides optimization for a limited set of queries. Local-as-View (LAV) is the best suited mediator approach for dynamic contexts as the Web [1]. Algorithms to answer queries using LAV views have a high complexity [65], and as in the Semantic Web context the queries may have a larger number of subgoals and the number of views may be huge, then its usability is limited.

5

SemLAV

5.1 Preliminaries

Mediators are components of the mediator-wrapper architecture [85]. They provide a uniform interface to autonomous and heterogeneous data sources. Mediators also rewrite an input query into queries against the data sources, and merge data collected from the selected sources. Wrappers are software components that assure the interoperability between sources and mediators by translating data collected from the sources into the schema and format understood by the mediators; the schema exposed by the wrappers is part of the schema exposed by its corresponding mediator.

The problem of processing a query Q over a set of heterogeneous data sources corresponds to answer Q using the instances of these sources. Although this problem has been extensively studied by the Database community [34], it has not been addressed for SPARQL queries. The following definitions are taken from Database existing solutions. Read the Section 2.2 if background on conjunctive queries is needed, and Section 4.2 if background on Data Integration is needed.

Definition 10 (LAV Integration System [48]). *A LAV integration system is a triple IS=$< G, S, M >$ where G is a global schema, S is a set of sources or source schema, and M is a set of views that map sources in S into the global schema G.*

For the rest of this part, we assume that views in M are limited to conjunctive queries. Both views and mediator queries are defined over predicates in G.

Theorem 5.1.1 (Number of Candidate Rewritings [1]). *Let N, O and M be the number of query subgoals, the maximal number of views subgoals, and the set of views, respectively. The number of candidate rewritings in the worst case is: $(O \times |M|)^N$.*

Theorem 5.1.2 (Complexity of Finding Rewritings [34]). *The problem of finding an equivalent rewriting is NP-complete.*

Consider the maximally-contained rewriting of a query Q (Definition 8, Section 4.2.2), Q', that uses as language, L, the union of conjunctive queries. View v can be used to answer query Q if there is one conjunctive query $r \in Q'$ such that v appears as the relation of one of r query subgoals. As $Q' \sqsubseteq Q$, then $r \sqsubseteq Q$. View v is called a relevant view atom. The next definition formalizes this notion.

Definition 11 (Relevant View Atom [26]). *A view atom v is relevant for a query atom g if one of its subgoals can play the role of g in the rewriting. To do that, several conditions must be satisfied: (1) the view subgoal should be over the same predicate as g, and (2) if g includes a distinguished variable of the query, then the corresponding variable in v must be a distinguished variable in the view definition.*

The concepts of relevant view and coverage have been widely used in the literature [26, 34]; nevertheless, they have been introduced in an informal way. The following definitions precise the properties that are assumed in this chapter.

Definition 12 (Relevant Views). *Let Q be a conjunctive query, $M = \{v_1, \ldots, v_m\}$ be a set of view definitions, and q be a query subgoal, i.e., $q \in body(Q)$. The set of relevant views for q corresponds to the set of relevant view atoms for the query subgoal q, i.e., $RV(M, q) = \{\tau(v) : v \in M \wedge w \in body(v) \wedge \psi(q) = \tau(w) \wedge (\forall x : x \in Vars(q) \wedge distinguished(x, Q) : distinguished(x, v))\}$[1]. The set of relevant views for Q corresponds to the views that are relevant for at least one query subgoal, i.e., $RV(M, Q) = \{\tau(v) : q \in body(Q) \wedge v \in M \wedge w \in body(v) \wedge \psi(q) = \tau(w) \wedge (\forall x : x \in Vars(q) \wedge distinguished(x, Q) : distinguished(x, v))\}$.*

1. $\psi(q)$ corresponds to the application of ψ to the variables of q (idem for $\tau(w)$).

Definition 13 (Coverage)**.** *Let Q be a conjunctive query, v be a view definition, q be a query subgoal, and w be a view subgoal. The predicate $covers(w, q)$ holds if and only if w can play the role of q in a query rewriting.*

We illustrate some of the given definitions for the LAV-based query rewriting approach using SPARQL queries. This will provide evidence of the approach limitations even for simple queries. In the following example, the global schema G is defined over the Berlin Benchmark [14] vocabulary. Consider a SPARQL query Q on G; Q has seven subgoals and returns information about products as shown in Listing 5.1. Listing 5.3 presents Q as a conjunctive query, where triple patterns are represented as query subgoals.

Listing 5.1 – SPARQL query Q

```
SELECT *
WHERE {
  ?X1 rdfs:label ?X2 .
  ?X1 rdfs:comment ?X3 .
  ?X1 bsbm:productPropertyTextual1 ?X8 .
  ?X1 bsbm:productPropertyTextual2 ?X9 .
  ?X1 bsbm:productPropertyTextual3 ?X10 .
  ?X1 bsbm:productPropertyNumeric1 ?X11 .
  ?X1 bsbm:productPropertyNumeric2 ?X12 .
}
```

Listing 5.2 – SPARQL View $s1$

```
SELECT *
WHERE {
  ?X1 rdfs:label ?X2 .
  ?X1 rdf:type ?X3 .
  ?X1 bsbm:productFeature ?X4 .
}
```

Listing 5.3 – Q expressed as a conjunctive query

```
Q(X1, X2, X3, X8, X9, X10, X11, X12) :- label(X1, X2), comment(X1, X3),
      productPropertyTextual1(X1, X8), productPropertyTextual2(X1, X9),
      productPropertyTextual3(X1, X10), productPropertyNumeric1(X1, X11),
      productPropertyNumeric2(X1, X12)
```

Listing 5.4 – Views s1-s10 from [21]

```
s1(X1,X2,X3,X4):-label(X1,X2),type(X1,X3),productfeature(X1,X4)
s2(X1,X2,X3):-type(X1,X2),productfeature(X1,X3)
s3(X1,X2,X3,X4):-producer(X1,X2),label(X2,X3),publisher(X1,X2),
      productfeature(X1,X4)
s4(X1,X2,X3):-productfeature(X1,X2),label(X2,X3)
s5(X1,X2,X3,X4,X5,X6,X7):-label(X1,X2),comment(X1,X3),producer(X1,X4),
      label(X4,X5),publisher(X1,X4),productpropertytextual1(X1,X6),
      productpropertynumeric1(X1,X7)
s6(X1,X2,X3,X4,X5):-label(X1,X2),product(X3,X1),price(X3,X4),vendor(X3,X5)
s7(X1,X2,X3,X4,X5,X6):-label(X1,X2),reviewfor(X3,X1),reviewer(X3,X4),
      name(X4,X5),title(X3,X6)
s9(X1,X2,X3,X4):-reviewfor(X1,X2),title(X1,X3),text(X1,X4)
s10(X1,X2,X3):-reviewfor(X1,X2),rating1(X1,X3)
s11(X1,X2,X3,X4,X5,X6,X7):-label(X1,X2),comment(X1,X3),producer(X1,X4),
      label(X4,X5),publisher(X1,X4),productpropertytextual2(X1,X6),
      productpropertynumeric2(X1,X7)
s12(X1,X2,X3,X4,X5,X6,X7):-label(X1,X2),comment(X1,X3),producer(X1,X4),
      label(X4,X5),publisher(X1,X4),productpropertytextual3(X1,X6),
      productpropertynumeric3(X1,X7)
s13(X1,X2,X3,X4,X5,X6,X7):-label(X1,X2),product(X3,X1),price(X3,X4),
      vendor(X3,X5),offerwebpage(X3,X6),homepage(X5,X7)
s14(X1,X2,X3,X4,X5,X6,X7):-label(X1,X2),product(X3,X1),price(X3,X4),
      vendor(X3,X5),deliverydays(X3,X6),validto(X3,X7)
s15(X1,X2,X3,X4,X5,X6,X7,X8,X9):-product(X1,X2),price(X1,X3),vendor(X1,X4),
      label(X4,X5),country(X4,X6),publisher(X1,X4),reviewfor(X7,X2),
      reviewer(X7,X8),name(X8,X9)
```

Consider M composed of 14 data sources defined as conjunctive views over the global schema G as in Listing 5.4; the Berlin Benchmark [14] vocabulary terms are represented as binary predicates in the conjunctive queries that define the data sources. Source *s1* can be defined as in Listing 5.2; note that we have done just a syntactic translation from this SPARQL query to the conjunctive query presented in Listing 5.4.

For instance, *s1* retrieves information about product type, label and product feature. The `rdfs:label` predicate is a *general predicate*. Commonly, general predicates are part of the definition of many data sources, and the number of rewritings of SPARQL queries that comprise triple patterns bound to general predicates can be very large. The general predicate `rdfs:label` in query Q can be mapped to views *s1, s3-s7, s11-s15*.

Listing 5.5 – A query rewriting for Q

```
r(X1,X2,X3,X8,X9,X10,X11,X12) :- s6(X1,X2,_0,_1,_2),
     s5(X1,_3,X3,_4,_5,_6,_7), s5(X1,_8,_9,_10,_11,X8,_12),
     s11(X1,_13,_14,_15,_16,X9,_17), s12(X1,_18,_19,_20,_21,X10,_22),
     s5(X1,_23,_24,_25,_26,_27,X11), s11(X1,_28,_29,_30,_31,_32,X12)
```

Listing 5.5 presents a query rewriting for Q, its subgoals cover each of the query subgoals of Q, e.g., $s6(X1, X2, \_0, \_1, \_2)$ covers the first query subgoal of Q, $label(X1, X2)$. $\psi(label(X1, X2)) = \tau(label(X1, X2))$; the mapping τ from view variables to rewriting variables is: $\tau(X1) = X1$, $\tau(X2) = X2$, $\tau(X3) = \_0$, $\tau(X4) = \_1$, $\tau(X5) = \_2$, and the mapping ψ from query variables to rewriting variables is: $\psi(Xi) = Xi$, for all Xi in the query head. Then, view $s6(X1, X2, \_0, \_1, \_2)$ is relevant for answering the first query subgoal of Q. Notice that the third, fourth and fifth projected variables of $s6$ correspond to existential variables because they are not relevant to cover the first query subgoal of Q with $s6$.

To illustrate how the number of rewritings for Q can be affected by the number of data sources that use the general predicate `rdfs:label`, we run the LAV query rewriter MCDSAT [10]. [2] First, if 14 data sources are considered, Q can be rewritten in 42 rewritings. For 28 data sources, there are 5,376 rewritings, and 1.12743e+10 rewritings are generated for 224 sources. [3] With one simple query, we can illustrate that the number of rewritings can be extremely large, being in the worst case exponential in the number of query subgoals and polynomial on the number of views. In addition to the problem of enumerating this large number of query rewritings, the time needed to evaluate them may be excessively large. Even using reasonable timeouts, only a small number of rewritings may be produced.

Table 5.1 shows the number of rewritings obtained by the state-of-the-art LAV rewriters GQR[44], MCDSAT[10] and MiniCon[65], when 224 views are considered for Q and timeouts are set up to 5, 10 and 20 minutes. Note that all these rewriters are able to produce only empty results or a small number of rewritings (up to 898,766 out of 1.12743e+10 $\approx$ 0.008 %).

In summary, even if the LAV approach constitutes a flexible approach to integrate data from heterogeneous data sources, query rewriting and processing tasks may be unfeasible in the context of SPARQL queries. Either the number of query rewritings is too large to be enumerated or executed in

2. MCDSAT [10] is the only query rewriting tool publicly available that counts the number of rewritings without having to enumerate all of them.

3. The 14 data sources setup is defined as in Listing 5.4, the one with 28 data sources has two views for each of the views in Listing 5.4, and the one with 224 sources has 16 views for each of the views in Listing 5.4

Table 5.1 – Number of rewritings obtained from the rewriters GQR, MCDSAT and MiniCon with timeouts of 5, 10 and 20 minutes. Using 224 views and query Q

Rewriter	5 minutes	10 minutes	20 minutes
GQR	0	0	0
MCDSAT	211,125	440,308	898,766
MiniCon	0	0	0

a reasonable time. To overcome these limitations and make feasible the LAV approach for SPARQL queries, we propose a novel approach named SemLAV. SemLAV identifies and ranks the relevant views of a query, and executes the query over the data collected from the relevant views; thus, SemLAV is able to output a high proportion of the answer in a short time.

5.2 The SemLAV Approach

SemLAV is a scalable LAV-based approach for processing SPARQL queries. It is able to produce answers even for SPARQL queries and integration systems with a large number of views and no statistics. SemLAV follows the traditional mediator-wrapper architecture [85]. Schemas exposed by the mediators and wrappers are expressed as RDF vocabularies. Given a SPARQL query Q over a global schema G and a set of sound views $M = \{v_1, \ldots, v_m\}$, SemLAV executes the original query Q rather than generating and executing rewritings as in traditional LAV approaches. SemLAV builds an instance of the global schema on-the-fly with data collected from the relevant views. The relevant views are considered in an order that enables to produce results as soon as the query Q is executed against this instance.

Contrary to traditional wrappers which populate structures that represent the heads of the corresponding views, SemLAV wrappers return RDF Graphs composed of the triples that match the triple patterns in the definition of the views. SemLAV wrappers could be more expensive in space than the traditional ones. Moreover, contrarily to existing mediator approaches, data is extracted from the sources and stored in the mediator, but only during query execution. However, they ensure that original queries are executable even for full SPARQL queries, and they make query execution dependent on the number of views rather than on the number of rewritings.

To illustrate the SemLAV approach, consider a SPARQL query Q with four subgoals given in Listing 5.6, and a set M of five views given in Listing 5.7.

Listing 5.6 – Products, features, and vendor of the offers

```
SELECT * WHERE {
    ?Offer bsbm:vendor ?Vendor .
    ?Vendor rdfs:label ?Label .
    ?Offer bsbm:product ?Product .
    ?Product bsbm:productFeature ?ProductFeature .
}
```

Listing 5.7 – Views that describe contents of five sources having data about products

```
v1(P,L,T,F):-label(P,L),type(P,T),productfeature(P,F)
v2(P,R,L,B,F):-producer(P,R),label(R,L),publisher(P,B),productfeature(P,F)
v3(P,L,O,R,V):-label(P,L),product(O,P),price(O,R),vendor(O,V)
v4(P,O,R,V,L,U,H):-product(O,P),price(O,R),vendor(O,V),label(V,L),offerwebpage(O,U),homepage(V,H)
v5(O,V,L,C):-vendor(O,V),label(V,L),country(V,C)
```

In the traditional LAV approach, 60 rewritings are generated, and the execution of these 60 rewritings produces all possible answers. [4] However, the generation and execution of the rewritings is time-consuming, and uses a non-negligible amount of memory to store data collected from views present in the rewritings. If there are not enough resources to execute all these rewritings, as many rewritings as possible will be executed. We apply a similar idea in SemLAV, if it is not possible to build the whole global schema instance to ensure a complete answer, then a partial instance will be built. The partial instance will include data collected from as many relevant views as the available resources allow, and if the relevant views are selective, the size of the partial instance should remain small and fit in memory.

The execution of the query over this partial schema instance will cover the results of executing a number of rewritings. The number of rewritings covered by the execution of Q over the partial schema instance could be exponential in the number of views included in the instance. Therefore, the size of the set of covered rewritings may be even greater than the number of rewritings executable in the same amount of time.

The order in which views are included in the partial global schema instance impacts the number of covered rewritings. Consider two different orders for including the views of the above example: v5, v1, v3, v2, v4 and v4, v2, v3, v1, v5. Table 5.2 considers partial global schema instances of different sizes. For each partial global schema instance, the included views and the number of covered rewritings are

4. Rewritings can be obtained with the Bucket algorithm given in Section 4.2.3, in this case all the queries in the Cartesian product of the buckets are valid rewritings because no contradictions among the predicates are attainable and all the variables in the views are distinguishable.

Table 5.2 – Impact of the different views ordering on the number of covered rewritings

# Included views (k)	Order One		Order Two	
	Included views (V_k)	# Covered rewritings	Included views (V_k)	# Covered rewritings
1	v5	0	v4	0
2	v5, v1	0	v4, v2	2
3	v5, v1, v3	6	v4, v2, v3	12
4	v5, v1, v3, v2	8	v4, v2, v3, v1	32
5	v5, v1, v3, v2, v4	60	v4, v2, v3, v1, v5	60

presented. Executing Q over the growing instances corresponds to the execution of a quite different number of rewritings. For instance, if only four views are included, one order corresponds to the execution of 32 rewritings while the another one corresponds to the execution of only eight rewritings. If all relevant views for query Q are included, then a complete answer is produced. However, if the number of relevant views is considerably large, it might be only possible to include k relevant views, V_k, in the global schema instance. As it has been shown in the previous example, the actual set of views, to be included in the global schema instance, determines the number of covered rewritings. With no knowledge about data distribution, we can only suppose that each rewriting has nearly the same chances of producing answers. Therefore, in order to increase the chances of obtaining answers from the global schema instance, we should include the set of k views that cover more query rewritings.

Maximal Coverage Problem (MaxCov). *Given an integer $k > 0$, a query Q on a global schema G, a set M of sound views over G, and a set R of conjunctive queries whose union is a maximally-contained rewriting of Q in M. The Maximal Coverage Problem is to find a subset V_k of M comprised of k relevant views for Q, $V_k \subseteq M \wedge (\forall v : v \in V_k : v \in RV(Q, M)) \wedge |V_k| = k$, such that the set of rewritings covered by V_k, $Coverage(V_k, R)$, is maximal for all subsets of M of size k, i.e., there is no other set of k views that can cover more rewritings than V_k. $Coverage(V_k, R)$ is defined as:*

$$Coverage(V_k, R) = \{r : r \in R \wedge (\forall p : p \in body(r) : p \in V_k)\} \tag{5.1}$$

The MaxCov problem has as input a solution to the Maximally-Contained Rewriting problem. Nevertheless, using this for building a MaxCov solution would be unreasonable since it makes the MaxCov solution at least as expensive as the rewriting generation. Instead of generating the rewrit-

ings, we define a formula that estimates the number of covered rewritings when Q is executed over a global schema instance that includes a set of views. It is the product of the number of ways each query subgoal can be covered by the set of views. For a query $Q(\bar{X}) \text{ :- } p_1(\bar{X}_1), \ldots p_n(\bar{X}_n)$ using only views in V_k this formula is expressed as:

$$NumberOfCoveredRewritings(Q, V_k) = \Pi_{1 \leq i \leq n} |Use(V_k, p_i(\bar{X}_i))|, \tag{5.2}$$

where $Use(V_k, p) = \Sigma_{v \in V_k} \Sigma_{w \in body(v) \wedge covers(w,p)} 1$. This formula computes the number of candidate rewritings, and it is also the exact number of covered rewritings when all the view variables are distinguished; this is because the coverage of each query subgoal by a given view can be considered in isolation, and queries composed by views that cover all the query subgoals are valid query rewritings. Otherwise, this expression is only an upper bound of the number of covered rewritings of Q with respect to V_k.

Consider the second proposed ordering of the views in the above example, the numbers of views in V_4 that cover each query subgoal are:

— two for the first query subgoal (v4 and v3),

— four for the second query subgoal (v4, v2, v3 and v1),

— two for the third query subgoal (v4 and v3), and

— two for the fourth query subgoal (v2 and v1).

Thus, the number of covered rewritings is 32 ($2 \times 4 \times 2 \times 2$).

Next, we detail a solution to the MaxCov problem under the assumption that views only contain distinguished variables.

5.2.1 The SemLAV Relevant View Selection and Ranking Algorithm

The relevant view selection and ranking algorithm is defined in Algorithm 3. This algorithm finds the views that cover each query subgoal (lines 2-13). This algorithm creates a bucket for each query subgoal q, where a bucket is a set of relevant views; this resembles the first step of the Bucket algorithm [50, 34] (lines 2-13). Additionally, the algorithm sorts the buckets views according to the number of covered subgoals (lines 14-17). Hence, the views that are more likely to contribute to the answer will be considered first.

Algorithm 3 The Relevant View Selection and Ranking

Require: Q : SPARQLQuery; M: set of View (defined as ConjunctiveQuery)
Ensure: *Buckets*: Predicate → list of View
1: **function** RELEVANTVIEWSELECTIONANDRANKING(Q, M)
2: **for all** $q \in body(Q)$ **do**
3: $b \leftarrow \emptyset$
4: **for all** $v \in M$ **do**
5: **for all** $w \in body(v)$ **do**
6: **if** There are mappings τ, ψ, such that $\psi(q) = \tau(w)$ **then**
7: $v_i \leftarrow \lambda(v)$ ▷ $\lambda(v)$ replaces all variables a_i in the head of v by $\tau(a_i)$
8: $insert(b, vi)$ ▷ add v_i to the bucket if it is not redundant
9: **end if**
10: **end for**
11: **end for**
12: $Buckets(q) \leftarrow$ b
13: **end for**
14: **for all** $q \in body(Q)$ **do**
15: $b \leftarrow Buckets(q)$
16: sortBucket(Buckets,b) ▷ MergeSort with key (#covered buckets,#view subgoals)
17: **end for**
18: **return** Buckets
19: **end function**

The mapping τ (line 6) relates view variables to query variables as stated in Definition 12.

The *sortBucket(buckets, b, q)* procedure (line 16) decreasingly sorts the views of bucket b according to the number of covered subgoals. Views covering the same number of subgoals are sorted decreasingly according to their number of subgoals. Intuitively, this second sort criterion prioritizes the more selective views, reducing the size of the global schema instance. The sorting is implemented as a classical *MergeSort* algorithm with a complexity of $O(|M| \times log(|M|))$.

Proposition 3. *The complexity of Algorithm 3 is* $Max(O(N \times |M| \times P), O(N \times |M| \times log(|M|)))$ *where* N *is the number of query subgoals,* M *is the set of views and* P *is the maximal number of view subgoals.*

To illustrate Algorithm 3, consider the SPARQL query Q and the previously defined views v1-v5. Algorithm 3 creates a bucket for each subgoal in Q as shown in Table 5.3a. For instance, the bucket of subgoal $vendor(O, V)$ contains v3, v4 and v5: all the views having a subgoal covering $vendor(O, V)$. The final output after executing the *sortBucket* procedure is described in Table 5.3b. Views v3 and v4 cover three subgoals, but since v4 definition has more subgoals, i.e., it is more selective, v4 is placed before v3 in all the buckets.

5.2.2 Global Schema Instance Construction and Query Execution

The global schema instance is constructed as described in Algorithm 4. Each bucket is considered as a stack of views, having on the top the view that covers more query subgoals (line 25). Iteratively, one view is popped from each bucket and its data is loaded into the instance (lines 28-42).

Table 5.3 – Buckets produced by Algorithm 3, included views (V_k) obtained by Algorithm 4, and the number of covered rewritings by V_k, for the query given in Listing 5.6

(a) Unsorted buckets

vendor(O,V)	label(V,L)	product(O,P)	productfeature(P,F)
v3(P,L,O,R,V)	v1(P,L,T,F)	v3(P,L,O,R,V)	v1(P,L,T,F)
v4(P,O,R,V,L,U,H)	v2(P,R,L,B,F)	v4(P,O,R,V,L,U,H)	v2(P,R,L,B,F)
v5(O,V,L,C)	v3(P,L,O,R,V)		
	v4(P,O,R,V,L,U,H)		
	v5(O,V,L,C)		

(b) Sorted buckets

vendor(O,V)	label(V,L)	product(O,P)	productfeature(P,F)
v4(P,O,R,V,L,U,H)	v4(P,O,R,V,L,U,H)	v4(P,O,R,V,L,U,H)	v2(P,R,L,B,F)
v3(P,L,O,R,V)	v3(P,L,O,R,V)	v3(P,L,O,R,V)	v1(P,L,T,F)
v5(O,V,L,C)	v2(P,R,L,B,F)		
	v1(P,L,T,F)		
	v5(O,V,L,C)		

(c) Included views

# Included views (k)	Included views (V_k)	# Covered rewritings
1	v4	$1 \times 1 \times 1 \times 0 = 0$
2	v4, v2	$1 \times 2 \times 1 \times 1 = 2$
3	v4, v2, v3	$2 \times 3 \times 2 \times 1 = 12$
4	v4, v2, v3, v1	$2 \times 4 \times 2 \times 2 = 32$
5	v4, v2, v3, v1, v5	$3 \times 5 \times 2 \times 2 = 60$

Table 5.3c shows how the number of covered rewritings increases as views are included into the global schema instance. Each V_k in this table is a solution to the MaxCov problem, i.e., the number of covered rewritings for each V_k is maximal. There are two possible options regarding query execution. Query can be executed each time a new view is included into the schema instance and partial results will be produced incrementally (line 34); or, it can be executed after including the k views (line 43). The first option prioritizes the time for obtaining the first answer, while the second one favors the total time to receive all the answers of Q over V_k. The first option produces results as soon as possible; however, in case of non-monotonic queries, i.e., queries where partial results may not be part of the query answer, this query processing approach should not be applied. Among non-monotonic queries, there are queries with modifiers like `ORDER BY` or constraints like a `FILTER` that includes the negation of a `bound` expression. The processing of non-monotonic queries requires all the relevant views to be included in the global schema instance in order to produce the same answer as it is produced using all the data accessible through the views.

Algorithm 4 The Global Schema Instance Construction and Query Execution

```
Require: Q : SPARQLQuery
Require: Buckets: Predicate → list of View        ▷ The buckets produced by Algorithm 3
Require: k : int
Ensure: A: set of Answer
20: function GRAPHINSTANCECONSTRUCTIONANDQUERYEXECUTION(Q, Buckets, k)
21:     Stacks : Predicate → stack of View
22:     V_k : set of View
23:     G : RDFGraph
24:     for all p ∈ domain(Buckets) do
25:         Stacks(p) ← toStack(Buckets(p))
26:     end for
27:     V_k, G ← ∅, ∅
28:     while (∃p| : ¬empty(Stacks(p))) ∧ |V_k| < k do
29:         for all p ∈ domain(Stacks) do
30:             if ¬empty(Stacks(p)) then
31:                 v ← pop(Stack(p))
32:                 if v ∉ V_k then
33:                     load v into G                    ▷ only if is not redundant
34:                     A ← A ∪ exec(Q, G)               ▷ Option 1: Execute Q after each successful load
35:                     V_k ← V_k ∪ {v}
36:                     if |V_k| = k then
37:                         break
38:                     end if
39:                 end if
40:             end if
41:         end for
42:     end while
43:     A ← exec(Q, G)                                   ▷ Option 2: execute before exit
44:     return A
45: end function
```

Proposition 4. *Considering conjunctive queries, the time complexity of Algorithm 4 in option 1 is $O(k \times N \times I)$, while the time complexity is $O(N \times I)$ for option 2. Where k is the number of relevant views included in the instance, N the number of query subgoals, and I is the size of the constructed global schema instance.*

5.2.3 The SemLAV Properties

Given a SPARQL query Q over a global schema G, a set M of views over G, the set RV of views in M relevant for Q, a set R of conjunctive queries whose union is a maximally-contained rewriting of Q using M, and V_k a solution to the MaxCov problem produced by SemLAV.

— *Answer Completeness:* If SemLAV executes Q over a global schema instance I that includes all the data collected from views in RV, it produces the complete answer. SemLAV outputs the same answers as a traditional rewriting-based query processing approach:

$$\bigcup_{r \in R} r(I(M)) = Q(\bigcup_{v \in RV} I(v)). \tag{5.3}$$

— *Effectiveness:* if SemLAV executes Q over a global schema instance that includes all the data collected from views in RV, it produces the complete answer, i.e., it is effective, and its *effec-*

tiveness is defined to be 1. If there are some constraints in time or space, V_k might be smaller than RV, and SemLAV *effectiveness* is defined as:

$$Effectiveness(V_k) = \frac{|Coverage(V_k, R)|}{|R|}. \tag{5.4}$$

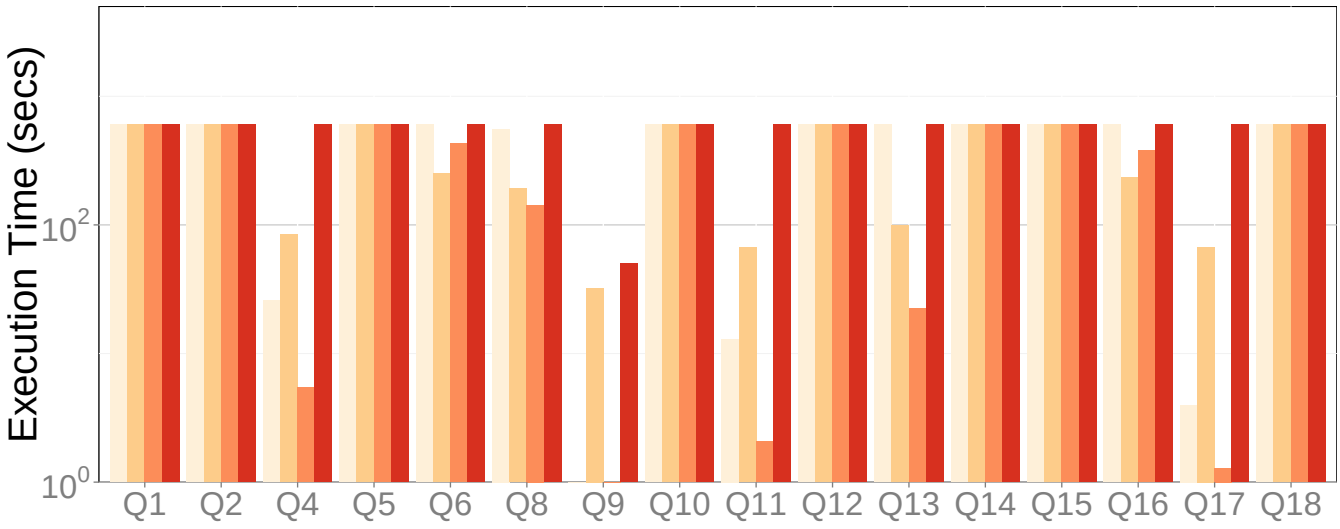
The value of effectiveness is a real number between 0 and 1. With no statistics about data distribution, it can be only supposed that each rewriting has nearly the same chances of producing answers. Therefore, this expression computes the chances of obtaining answers when Q is executed against a (partial) global schema instance that contains the data available through views in V_k. In order to answer Q as fast as possible, and reduce its evaluation cost, it is desirable for the effectiveness of V_k to be as high as possible, for all k.

— *Execution Time depends on $|RV|$:* The load and execution time of SemLAV linearly depends on the size of the views included in the global schema instance.

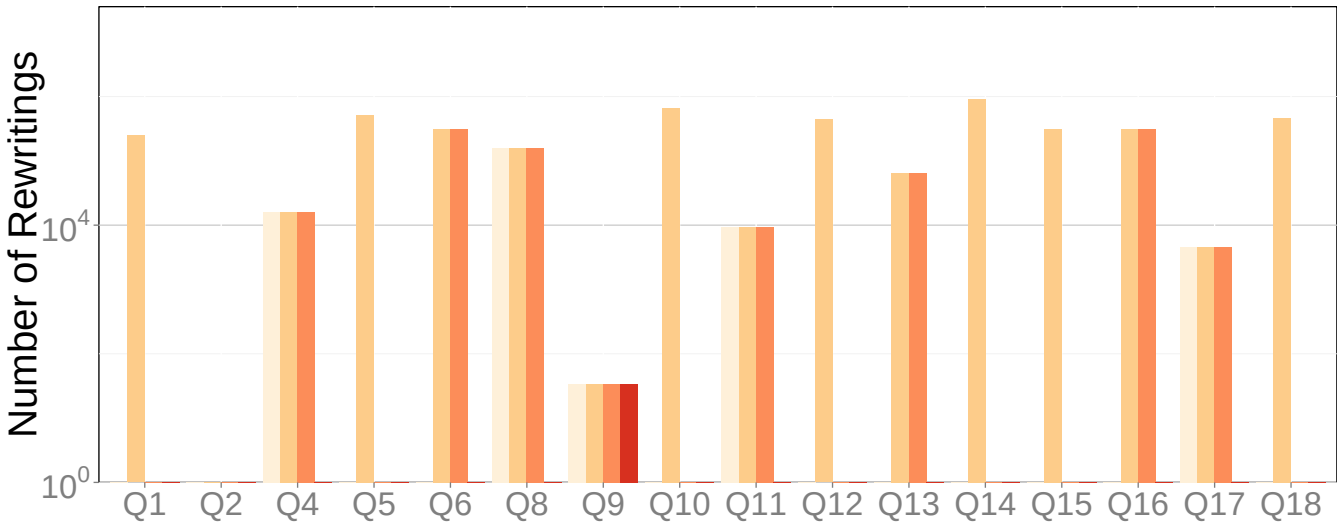
— *No memory blocking:* SemLAV guarantees to obtain a complete answer when $\bigcup_{v \in RV} I(v)$ fits into memory. If not, it is necessary to divide the set RV of relevant views into several subsets RV_i, such that each subset fits into memory and for any rewriting $r \in R$ all views $v \in body(r)$ are contained in one of these subsets.

5.3 Experimental Evaluation

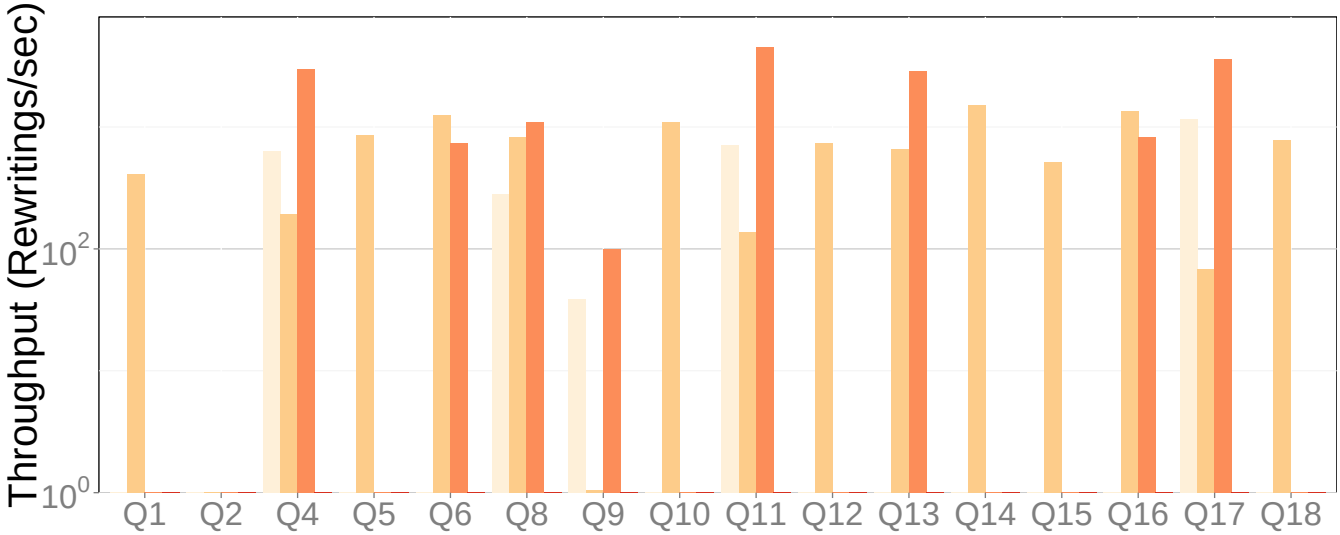
We compare the SemLAV approach with a traditional rewriting-based approach and analyze the SemLAV effectiveness, memory consumption, and throughput. In order to decide which rewriting engine will be used to compare with SemLAV, we run some preliminary experiments to compare existing state-of-the-art rewriting engines. We consider GQR [44], MCDSAT [10], MiniCon [65], and SSDSAT [40]. We execute these engines for 10 minutes and measure execution time in seconds and the number of rewritings generated by each engine. Additionally, we use these values to compute the throughput; throughput corresponds to the number of rewritings obtained per second. Figures 5.1a, 5.1b and 5.1c present the execution time, number of obtained rewriting and the throughput respectively. The GQR performance is good when the number of query rewritings is low and the views cover few query subgoals, and it outperforms some of the other engines. That is, this situation allows to speed up the preprocessing time consumed by GQR to build the structures



(a) Execution Time (secs)



(b) Number of Rewritings



(c) Rewritings Throughput

Figure 5.1 – Comparison of state-of-the-art LAV rewriting engines for 16 queries without existential variables and 476 views from our experimental setup. Studied engines are: GQR ( ), MCDSAT ( ), MiniCon ( ) and SSDSAT ( )

Table 5.4 – Queries and their answer size, number of subgoals, number of rewritings, and views size

(a) Query information

Query	Answer Size	# Subgoals	# Rewritings
Q1	6.68E+07	5	2.04E+10
Q2	5.99E+05	12	1.57E+24
Q4	2.87E+02	2	1.62E+04
Q5	5.64E+05	4	7.48E+07
Q6	1.97E+05	3	3.14E+05
Q8	5.64E+05	3	1.57E+05
Q9	2.82E+04	1	3.40E+01
Q10	2.99E+06	3	4.40E+06
Q11	2.99E+06	2	9.25E+03
Q12	5.99E+05	4	1.50E+09
Q13	5.99E+05	2	6.47E+04
Q14	5.64E+05	3	2.52E+06
Q15	2.82E+05	5	2.04E+10
Q16	2.82E+05	3	3.14E+05
Q17	1.97E+05	2	4.62E+03
Q18	5.64E+05	4	1.20E+09

(b) Views size

Views	# Triples
V1-V34	201,250
V35-V68	153,523
V69-V102	53,370
V103-V136	26,572
V137-V170	5,402
V171-V204	66,047
V205-V238	40,146
V239-V272	113,756
V273-V306	24,891
V307-V340	11,594
V341-V374	5,402
V375-V408	5,402
V409-V442	78,594
V443-V476	99,237
V477-V510	1,087,281

required to generate the query rewritings. The MCDSAT performance is good in a larger number of queries; it can produce rewritings for more queries than the other engines, particularly in queries with a large number of triple patterns and in presence of general predicates. However, MCDSAT does not outperform the other engines when they are able to produce rewritings. This is because of the overhead incurred by MCDSAT by translating the problem into a logical theory to be solved by a SAT solver. The MiniCon performance is pretty good in general, but it only produces query rewritings when the number of rewritings is relatively small. Finally, SSDSAT is able to handle constants; however, this feature severely impacts its performance, being able to produce rewritings only for simple cases.

5.3.1 Experimental Hypotheses

The hypotheses of our experiments are:

— SemLAV loads the more relevant views of a query first, the SemLAV effectiveness should be considerably high and should produce more answers than the rest of the engines in the same amount of time.

— SemLAV builds a global schema instance using data collected from the relevant views, SemLAV

may consume more space than a traditional rewriting-based approach.

— SemLAV produces results incrementally, it is able to produce answers sooner than a traditional rewriting-based approach.

5.3.2 Experimental Configuration

The Berlin SPARQL Benchmark (BSBM) [14] is used to generate a dataset of 10,000,736 triples using a scale factor of 28,211 products. Additionally, third-party queries and views are used to provide an unbiased evaluation of our approach. In our experiments, the goal is to study SemLAV as a solution to the MaxCov problem, and we compute the number of rewritings generated by three state-of-the-art query rewriters. From the 18 queries and 10 views defined in [21], we leave out the ones using constants (literals) because the state-of-the-art query rewriters are unable to handle constants either in the query or in the views. In total, we use 16 out of 18 queries and nine out of 10 the defined views. The query triple patterns can be grouped into chained connected star-shaped subqueries, that have between one and twelve subgoals with only distinguished variables, i.e., queries are free of existential variable. We define five additional views to cover all the predicates in the queries. From these 14 views, we produce 476 views by horizontally partitioning each original view into 34 parts, such that each part produces 1/34 of the answers given by the original view.

Queries and views are described in Tables 5.4a and 5.4b. The size of the complete answer is computed by including all the views into an RDF-Store (Jena) and executing the queries against this centralized RDF dataset.

We implement wrappers as simple file readers. For executing rewritings, we use one named graph per subgoal as done in [47]. The Jena 2.7.4 [5] library with main memory setup is used to store and query the graphs. The SemLAV algorithms are implemented in Java, using different threads for bucket construction, view inclusion and query execution to improve performance. The implementation is available in the project website [6].

5.3.3 Experimental Results

The analysis of our results focus on three main aspects: the SemLAV effectiveness, memory consumption and throughput.

5. http://jena.apache.org/
6. https://sites.google.com/site/semanticlav/

Table 5.5 – The SemLAV Effectiveness. For 10 minutes of execution, we report the number of relevant views included in the global schema instance, the number of covered rewritings and the achieved effectiveness. Effectiveness values higher than 0.5 are shown in **bold**

Query	Included Views / # Relevant Views	# Covered rewritings / # Rewritings	Effectiveness
Q1	30 / 408	2.28E+06 / 2.04E+10	0.000112
Q2	194 / 408	2.05E+23 / 1.57E+24	0.130135
Q4	156 / 374	8.77E+03 / 1.62E+04	**0.542017**
Q5	52 / 374	3.13E+06 / 7.48E+07	0.041770
Q6	44 / 136	2.13E+04 / 3.14E+05	0.067728
Q8	81 / 136	9.36E+04 / 1.57E+05	**0.595588**
Q9	34 / 34	3.40E+01 / 3.40E+01	**1.000000**
Q10	88 / 408	3.20E+05 / 4.40E+06	0.072766
Q11	77 / 136	5.24E+03 / 9.25E+03	**0.566176**
Q12	238 / 408	7.70E+08 / 1.50E+09	**0.514286**
Q13	245 / 408	4.26E+04 / 6.47E+04	**0.657563**
Q14	46 / 272	1.22E+04 / 2.52E+06	0.004837
Q15	70 / 442	5.12E+08 / 2.04E+10	0.025144
Q16	82 / 136	1.90E+05 / 3.14E+05	**0.602941**
Q17	56 / 136	1.90E+03 / 4.62E+03	0.411765
Q18	23 / 374	2.80E+05 / 1.20E+09	0.000234

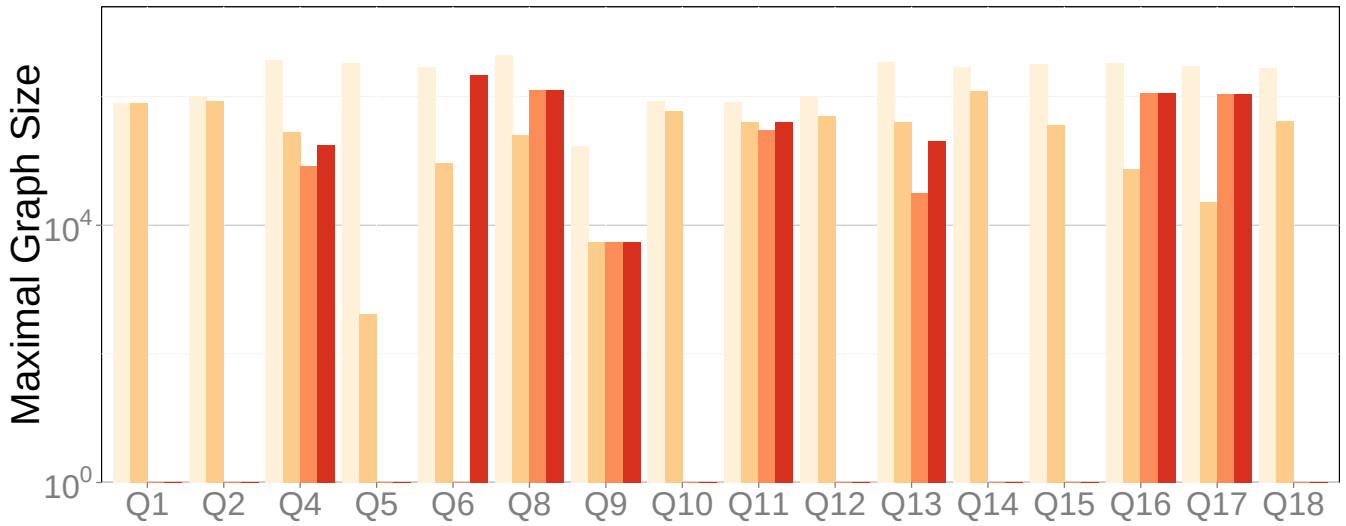


Figure 5.2 – Maximal Graph Size during query execution for SemLAV ( ), MCDSAT ( ), GQR ( ) and MiniCon ( ) approaches

To demonstrate the SemLAV effectiveness, we execute SemLAV with a timeout of 10 minutes. During this execution, the SemLAV algorithms select and include a subset of the relevant views; this set corresponds to V_k as a solution to the MaxCov problem. Then, we use these views to compute the number of covered rewritings using the formula given in Section 5.2. Table 5.5 shows the number of relevant views considered by SemLAV, the covered rewritings and the achieved effectiveness. Effectiveness is greater than or equal to 0.5 (out of 1) for almost half of the queries. SemLAV maximizes the number of covered rewritings by considering views that cover more subgoals first.

The observed results confirm that the SemLAV effectiveness is considerably high. Effectiveness depends on the number of relevant views, but this number is bounded to the number of relevant views

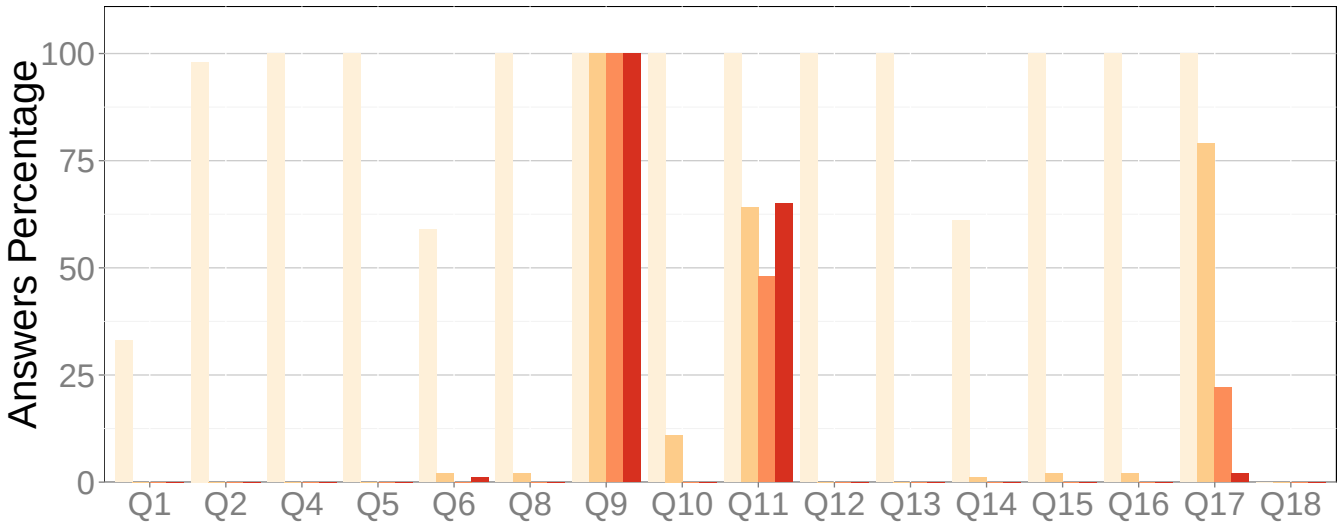


Figure 5.3 – Answer Percentage obtained by SemLAV ( ), MCDSAT ( ), GQR ( ) and MiniCon ( )

that can be stored in memory. As expected, the SemLAV approach could require more space than the traditional rewriting-based approach. SemLAV builds a global schema instance that includes all the relevant views in V_k, whereas a traditional rewriting-based approach includes only the views in one rewriting at the time. Figure 5.2 shows the maximal graph size in both approaches. SemLAV can use up to 129 times more memory than the traditional rewriting-based approach (for Q17). SemLAV can use less memory than the traditional rewriting-based approach (for Q1) for relevant views with overlapped data.

We calculate the throughput as the number of obtained answers divided by the total execution time. For SemLAV, this time includes view selection and ranking, contacting data sources using the wrappers, including data into the global schema instance, and query execution time. For the traditional rewriting-based approach, this time includes rewriting time, instead of view selection and ranking. Table A.1 in Appendix A.1 presents the complete results of the experiments, they include the number of answers, execution time, number of times the query is executed and throughput. Notice that SemLAV executes the query whenever a new relevant view has been included in the global schema instance and the query execution thread is active.

Figures 5.3 and 5.4 show an impressive difference in the answer percentage and throughput, e.g., for Q1 SemLAV produces 37,350.1 answers/sec, while the other approach produces up to 0.5 answers/sec. This huge difference is caused by the differences between the complexity of the rewriting generation and the SemLAV view selection and ranking algorithm, and between the number of

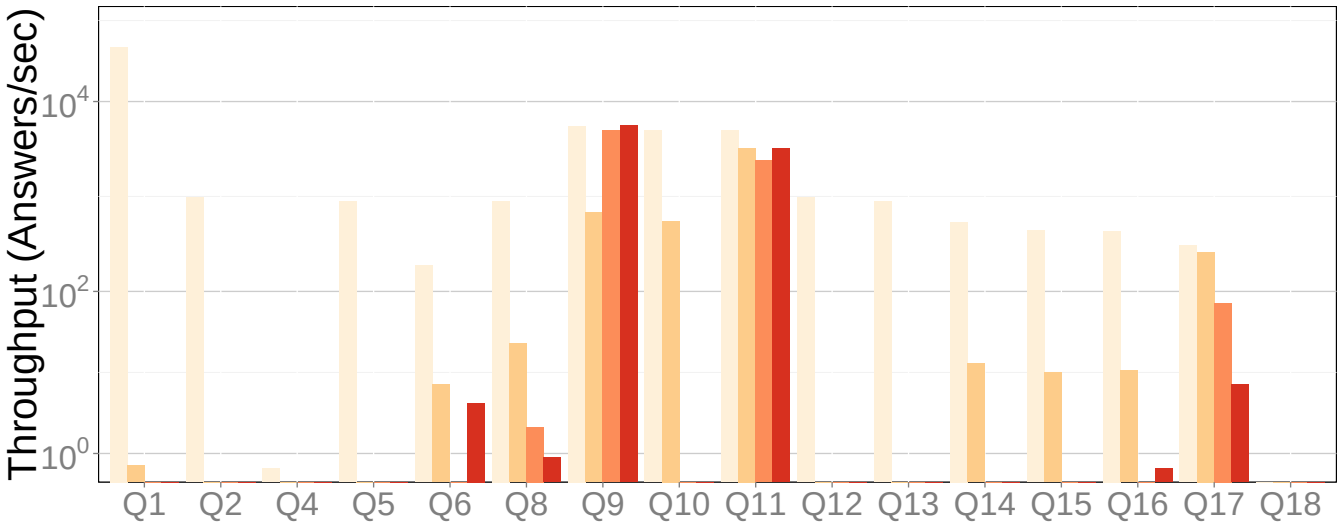


Figure 5.4 – Throughput of SemLAV ( ), MCDSAT ( ), GQR ( ) and MiniCon ( )

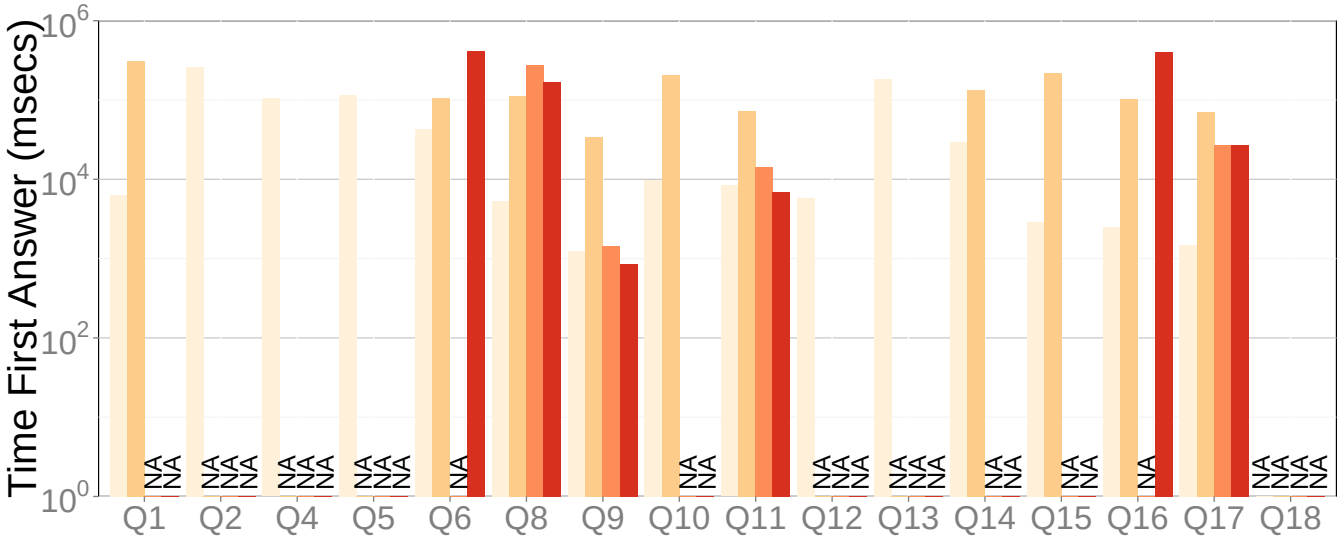


Figure 5.5 – Time of the First Answer (msec) of SemLAV ( ), MCDSAT ( ), GQR ( ) and MiniCon ( ). "NA" indicates that the approach did not produce answers for that query

rewritings and number of relevant views. This makes possible to generate answers sooner.

Figure 5.5 shows the time for the first answer (TFA); TFA is impacted by executing the query as soon as possible, according to option 1 given in Algorithm 4. Only for query $Q18$ SemLAV does not produce any answer in 10 minutes. This is because the views included in the global schema instance are large (around one million triples per view) and do not contribute to the answer; consequently, almost all the execution time is spent in transferring data from the relevant views. SemLAV produces answers sooner in all the other cases. Moreover, SemLAV also achieves complete answer in 11 of 16 queries in only 10 minutes.

In summary, the results show that SemLAV is effective and efficient and produces more answers

sooner than a traditional rewriting-based approach. SemLAV makes the LAV approach feasible for processing SPARQL queries.

5.4 Conclusions and Future Work

We have presented SemLAV, a Local-As-View mediation technique that allows to perform SPARQL queries over views without facing problems of NP-completeness, exponential number of rewritings or restriction to conjunctive SPARQL queries. This is obtained at the price of including relevant views into a global schema instance which is space consuming. However, we demonstrated that, even if only a subset of relevant views is included, we obtain more results than traditional rewriting-based techniques. Chances of producing results are higher, if the number of covered rewritings is maximized as defined in the MaxCov problem. We proved that our ranking strategy maximizes the number of covered rewritings.

SemLAV opens a new way to execute SPARQL queries for LAV mediators that is tractable. As perspectives, the performance of SemLAV can be greatly improved by parallelizing the inclusion of views. Currently, SemLAV includes views sequentially due to Jena restrictions. If views were included in parallel, time to get first results may be greatly improved. This perspective work has been partially addressed in [28], in this work parallel loading of views has been simulated by loading views in blocks, and loading blocks of different views. Nevertheless, a real parallel implementation of view loading may provide even better results.

Additionally, the strategy of producing results as soon as possible, can deteriorate the overall throughput. If users want to improve overall throughput, then the query should be executed once after all the views in V_k have been included. Moreover, query execution may be done in an incremental way, saving intermediate results for future query executions, and reducing the overheat of executing the same query several times. It could be also interesting to design an execution strategy where SemLAV would execute under constrained space. In this case, the problem would be to find the minimum set of relevant views that would fit in the available space and produce the maximal number of answers.

Answering SPARQL Queries against Federations with Replicated Fragments

6

Introduction

SPARQL endpoints enable to consume RDF data exploiting the *expressiveness* of the SPARQL query language. Nevertheless, recent studies reveal that existing public SPARQL endpoints main limitation is availability [8].

In distributed databases [62], a common practice to overcome availability problems is to replicate data near data consumers. Replication can be achieved by complete dataset replication, e.g., the LOD cloud cache endpoint [1] exposes data from several datasets present in the LOD cloud. But replication can be also achieved with a finer granularity, i.e., replication of the portions of the datasets that are relevant, e.g., in [39] users replicate only the fragments of data that they want to modify to improve their data quality.

RDF data consumers can replicate *subsets* of RDF datasets or *replicated fragments*, and make them accessible through SPARQL endpoints. This will provide the support for an efficient RDF data re-organization according to the *needs* and *computational resource capacity* of data consumers, while these data can be still *accessed* using SPARQL endpoints. Unfortunately, although SPARQL endpoints can *transparently* access replicated fragments, as well as maintain their *consistency* [39], federated query engines are not tailored to exploit the benefits of replicated fragments.

Federated SPARQL engines [2], [11], [32], [68], [74] allow data consumers to execute SPARQL

1. http://lod2.openlinksw.com/sparql, November, 2015.

queries against a federation of SPARQL endpoints. However, these engines are just designed to select the SPARQL endpoints that ensure both answer production and an efficient execution of the query. In presence of replication, existing federated query engines may retrieve data from every relevant endpoint, and transfer a large number of tuples that trigger many requests to the endpoints. Thus, federated query engines may exhibit poor *performance* while *availability* of the selected SPARQL endpoints is negatively impacted.

Although the problem of managing RDF data *overlapping* during federated query processing has been addressed in [38], [70], the problem of managing *replication* in a federation of RDF datasets still remains open. DAW [70] is able to detect overlapping between datasets and optimize source selection based on that. However, because DAW is not designed to manage data replication, there is no support for explicitly define and use replicated fragments. In consequence, DAW may select redundant data sources and generate a high number of transferred tuples as we will report in our experiments.

We build a replication-aware SPARQL federated query engine by integrating into state-of-the art federated query engines FedX [74] and ANAPSID [2], a source selection strategy called Fedra that solves the source selection problem with fragment replication (SSP-FR). For a given set of SPARQL endpoints with replicated fragments and a SPARQL query, the problem is to minimize the transferred data from endpoints to the federated query engines, while preserving answer completeness and reducing data redundancy.

We empirically study federated query engines FedX and ANAPSID extended with Fedra and DAW on synthetic and real datasets. The results suggest that Fedra efficiently reduces the number of transferred tuples and data redundancy.

This part is organized as follows. Chapter 7 presents related works, while chapter 8 presents Fedra. First, Section 8.1 describes background and motivations. Section 8.2 defines replicated fragments and presents the source selection problem for fragment replication. Section 8.3 presents the Fedra source selection algorithm. Section 8.4 reports our experimental results. Finally, conclusions and future works are outlined in Section 8.5.

7

State of the Art

7.1 Distributed Database Query Processing

Distributed query processing has been widely studied in databases [45]. A generic layering schema of distributed query processing is presented by Özsu and Valduriez in [62]. Figure 7.1 presents this schema. It is composed by four layers: query decomposition, data localization, global optimization and distributed execution. The query decomposition layer translates the calculus query into an algebraic query that corresponds to a "good" translation, this mean that rules that been applied to avoid typical bad algebraic queries. The data localization layer transforms the query on relations into a query on fragments. Fragments are disjoint subsets of relations whose precise location is stored at the fragment schema, if possible any further transformation to avoid bad algebraic plans is performed. The global optimization layer transforms the query on fragments into an optimized query on fragments or distributed query execution plan. This transformation explores possible operator orders and uses fragment sizes and allocation to choose the "best". Actual explorations consider only a subset of possible orders to keep their complexity low. And allocation depends on the possible multiple replicas that may exist for each fragment. The distributed execution layer is performed by each site having fragments in the distributed query execution plan. They may perform local

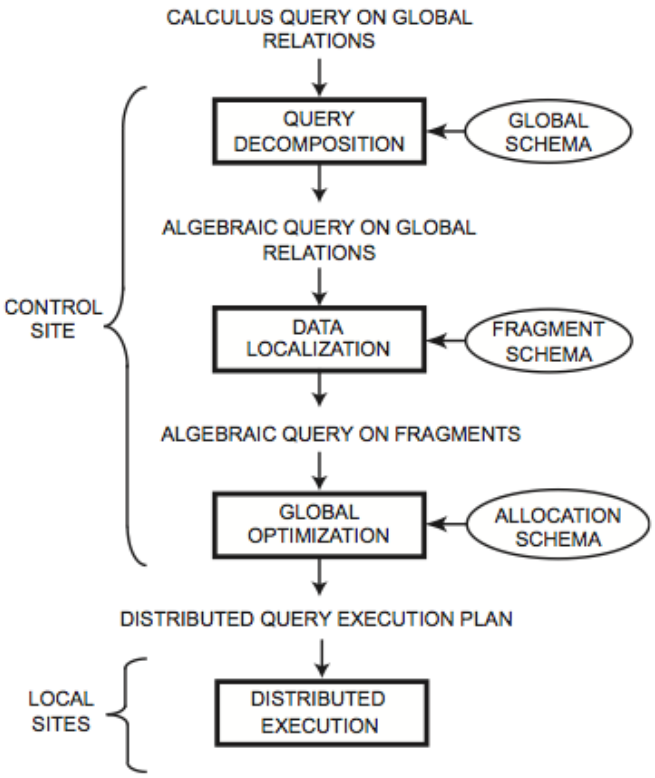


Figure 7.1 – Distributed Query Processing, Figure 6.3 at [62]

optimizations to the plan they receive and choose the physical operators to use. Fragmentation and replication contribute to improve the database availability, scalability and reliability; but it incurs in some additional cost to keep the replicas up to date. Data may be fragmented horizontally or vertically. Horizontal fragmentation distributes the tuples of a relation in different tables, while vertical fragmentation splits the tuples of a relation by attribute and places subtuples in different tables. Fragmentation contributes to decrease the cost of replication, copying only a subset of the relation tuples or attributes. If the database has been fragmented, fragments should be allocated near the final users in order to obtain better performance, i.e., reductions on stored data and network delays may be achieved if data is stored where it is needed. Data allocation requires knowledge about the queries that are going to be posed in the different sites or a way to predict them. Distributed database fragments are disjoint portions of the database, even if one fragment may be replicated in several sites it is clear and easy to determine which fragments are required in order to execute a query. Objects stored in a distributed database should converge to the same state. This can be achieved using locks that prevent databases of reaching inconsistent states, or using timestamps that contribute to roll the database back to the last consistent state. Data fragmentation is tailored for representative queries; fragments are smartly allocated and replicated across servers for balancing

workload and reducing size of intermediate results.

Linked Data [13] is intrinsically a federation of autonomous participants where federated queries are unknown to a single participant, and a tight coordination of data providers is difficult to achieve. Consequently, federated query engines cannot rely on properties ensured by a distributed database allocation algorithm. The challenge faced in this second part of the thesis is given a set of fragments replicated in a federation of SPARQL endpoints, that may or not overlap, make the best use of these fragments to evaluate subqueries locally, and consequently reduce the number of transferred tuples from the endpoints to the federated query engine.

7.2 Linked Data Query Processing

Görlitz and Staab summarize in [31] the three main approaches to query Linked Open Data:

— Central repository: all data are retrieved and stored in a central data store. Central indexes are used, and contact with original data sources is lost. Then, query execution can be properly optimized, but it may be needed to retrieve data periodically to ensure that the data is up to date. Data retrieving may be done using a dump file if available or crawling the RDF data.

— Explorative query processing: links present in the query are used to retrieve data and links to other sources that may have relevant data for the query. The query execution is done on the actual data, then it is always up to date, but it may produce incomplete answers, and the order in which links are explored may lead to different answers.

— Data source federation: query execution is performed at the sources, but indexes are keep in the federation to optimize query execution. Then, data is up to date, storage space for indexes is limited, and indexes information, if they are out of date, may lead to less efficient plans rather than to incomplete answers.

This last approach, federated query processing, is the one that presents more advantages, however there are many challenges to consider as keeping up to date indexes that can be used to optimize query processing with a bounded size and complexity of computation, and producing optimized plans that reduce the number of sources used, and the size of transferred data. This approach, and some of its exemplars are presented in the next sections.

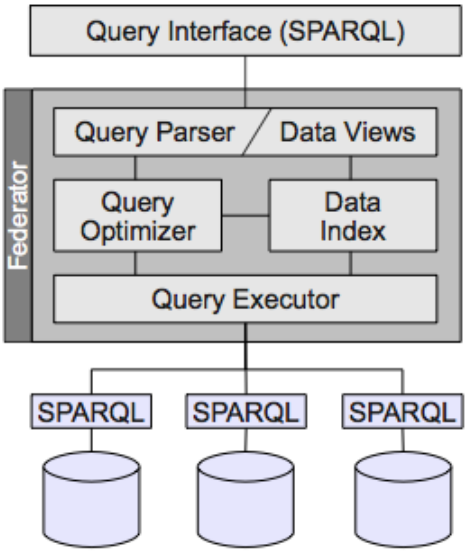


Figure 7.2 – Generic Federation Infrastructure, Figure 3 at [31]

7.3 Federated Query Processing

Linked open data sources share links between them. These links allow users to formulate queries that potentially use several data sources. Integrating sources from several sources can be done using Mediators and Wrappers architecture as discussed in Chapter 4. In the case of federations of SPARQL endpoints, the mediator may be simplified as many sources share ontologies and links among sources are already provided by them.

Figure 7.2 presents the main components in federated query processing. Each source provides access to data through a SPARQL endpoint, i.e., RESTful services that accept SPARQL queries over HTTP [7]. User queries, written in SPARQL, are decomposed into subqueries that are sent to the sources and the query executor is in charge of using source answers to produce the query answers. Indexes are used to determine the sources where subqueries should be evaluated, and the order of subqueries in the execution plan. Joins may be executed at the sources using SERVICE clauses, but it may require user privileges. Federated query engines may provide different join implementations, like hash joins, nested loop joins or bind joins.

7.4 Federated Query Processing Engines

Federated query engines are query processing engines, that given a query, are capable of retrieving data from (several) relevant sources, and producing the query answers. Federated query engines may use the SPARQL federation extension [6] and allow the user to specify which sources should be used

to execute subqueries, or may use a catalog of sources, and decompose the query into subqueries and assign to each subquery the endpoints where it should be executed as introduced in Section 7.3.

ARQ[1] and SPARQL-DQP [7] are examples of engines that use SERVICE clauses from the SPARQL federation extension. SPLENDID [32], and DARQ [68] are examples of engines that are able to decompose queries into subqueries and assign to these subqueries the endpoints where they should be executed. Finally, FedX [74] and ANAPSID [2] are both able to process queries with SERVICE clauses, and able to decompose queries into subqueries and determine the endpoints where these subqueries are to be evaluated.

Consider the query *Find French directors and their film genres*, as in query *Q* (Listing 7.1).

Listing 7.1 – SPARQL query Q

```
select distinct ?director ?genre where {
  ?director dbo:nationality dbr:France .
  ?film dbo:director ?director .
  ?movie owl:sameAs ?film .
  ?movie linkedmdb:genre ?genre }
```

Table 7.1 – *Federation1* and *Federation2* endpoints that have triples with predicates in the query *Q* (Listing 7.1)

Q triple pattern		Federation1	Federation2
tp_1	?director dbo:nationality dbr:France	E1	E3
tp_2	?film dbo:director ?director	E1	E4
tp_3	?movie owl:sameAs ?film	E1, E2	E3
tp_4	?movie linkedmdb:genre ?genre	E2	E4

And two different federations, *Federation1* and *Federation2*. Each federation has two SPARQL endpoints, *Federation1* has endpoints *E1* and *E2*, and *Federation2* has endpoints *E3* and *E4*. Table 7.1 presents the endpoints that have triples with predicates in each of the query triple patterns. An important difference among these federations is that while in *Federation1* the same endpoint, *E1*, has triples relevant for the triple patterns tp_1 and tp_2, connected by variable *?director*, in *Federation2* no endpoint has triples relevant for two triple patterns connected by a variable.

Listing 7.2 – SPARQL query Q to be executed against Federation1 (Table 7.1)

```
select distinct ?director ?genre where {
  SERVICE <http://E1/sparql> {
    ?director dbo:nationality dbr:France .
    ?film dbo:director ?director
```

1. https://jena.apache.org/documentation/query/index.html

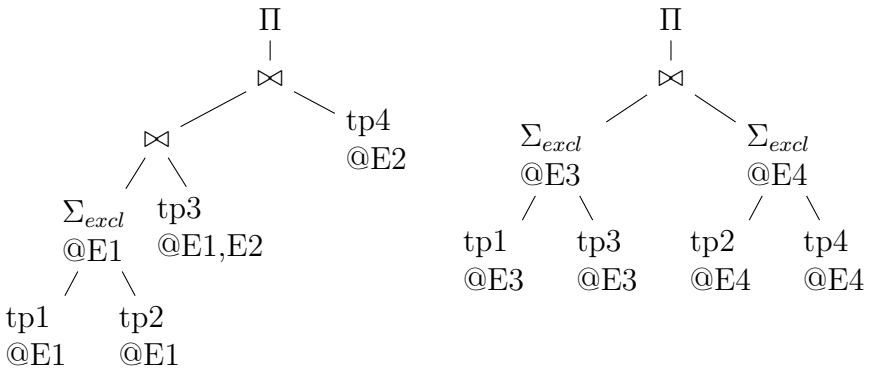
```
  } .
  SERVICE <http://E2/sparql> {
    ?movie owl:sameAs ?film .
    ?movie linkedmdb:genre ?genre
  }
}
```

To execute this query using the SPARQL federation extension [6], the user should specify where to execute each triple pattern using SERVICE clauses as depicted in Listing 7.2. Advanced users that know quite well the data sources, may be able to write this kind of query, but for users less familiar with the data sources or federations with a high number of data sources or dynamic federations, it may be too challenging for the user to choose where to execute each triple pattern. Asking queries like the one given in Listing 7.1 promotes Linked Data flexibility [31].

FedX [74] and ANAPSID [2, 60], state-of-art query engines for federations of SPARQL endpoints, are detailed in the following sections.

7.4.1 FedX

FedX [74] is a federated query engine built on top of the Sesame framework [18]. FedX keeps a catalog with the available sources, and the mappings between RDF terms and sources are built during query execution when they are needed. FedX source selection depends solely on `ASK` queries that are sent during query execution to the sources to determine if the can provide triples for a given query triple pattern. `ASK` query results can be stored in a cache for future use. For each triple pattern, it is determined if its relevant data is available in one or several sources. If data is available in exactly one endpoint, it is said that the triple pattern can be *exclusively* evaluated in that source. All the triple patterns that can be exclusively evaluated in one source can be group together in an *exclusive group*. Triple patterns that do not belong to an exclusive group are sent individually to all the endpoints that provide data for them. For query Q (Listing 7.1) and *Federation1* (Table 7.1), FedX builds an exclusive group composed of tp_1 and tp_2 to be sent to *E1*, and sends tp_3 individually to both *E1* and *E2*. Triple patterns are sorted in joins using a cost estimation heuristic, that takes into account their number of unbound variables. Triple patterns with the least number of unbound variables are evaluated first, because they are likely to incur in the smallest number of transferred tuples from endpoints to the query engine. Joins are combined in left-linear plans. Joins are evaluated in a block nested loop fashion, i.e., as distributed semi-joins, and intermediate results from inner to outer



(a) Federation1 (b) Federation2

Figure 7.3 – FedX execution plans for query Q (Listing 7.1) and federations *Federation1* and *Federation2* (Table 7.1)

operands are passed in blocks, and this reduces the number of requests sent to the endpoints by a factor equal to the block size.

Table 7.2 – Positive impact of the use of exclusive groups (EG) on the number of transferred tuples (TT) in Federation1 (Table 7.1) and Q (Listing 7.1)

With EG		Without EG	
Executed Subquery	# TT	Executed Subquery	# TT
tp_1 . tp_2	141	tp_1	1,700
tp_1 . tp_2 . tp_3	144	tp_1 . tp_2	1,841
tp_1 . tp_2 . tp_3 . tp_4	145	tp_1 . tp_2 . tp_3	1,844
		tp_1 . tp_2 . tp_3 . tp_4	1,845

For Q and *Federation1*, FedX builds the plan given in Figure 7.3a. Exclusive group composed of tp_1 and tp_2 is to be evaluated first, then tp_3 and finally tp_4. tp_3 should be evaluated in second place because having evaluated the exclusive group, values for variable $?film$ are available when tp_3 shall be evaluated, then tp_3 is likely to be less expensive than tp_4. To show the positive impact of the exclusive groups on the number of transferred tuples during FedX execution [2], we setup two Virtuoso7.2.1 endpoints, and populate them using data from DBpedia [3] and LinkedMDB [4]. The number of transferred tuples using exclusive group and without exclusive groups are presented in Table 7.2, using exclusive groups reduces the number of transferred tuples by one order of magnitude.

It is important to notice that FedX will group triple patterns that do not have any variable in common into an *exclusive group*, if they are exclusively provided by one endpoint in order to reduce the number of requests, but doing so can highly increase the number of transferred tuples. Moreover

2. FedX3.1 was used for the execution
3. DBpedia3.9 subset as in FedBench [73]
4. Version from January 19th, 2010

Table 7.3 – Negative impact of the use of exclusive groups (EG) on the number of transferred tuples (TT) in Federation2 (Table 7.1) and Q (Listing 7.1)

With EG		Without EG	
Executed Subquery	# TT	Executed Subquery	# TT
tp_1 . tp_3	299,978,600	tp_1	1,700
tp_1 . tp_3 . tp_2 . tp_4	299,978,601	tp_1 . tp_2	1,841
		tp_1 . tp_2 . tp_3	1,844
		tp_1 . tp_2 . tp_3 . tp_4	1,845

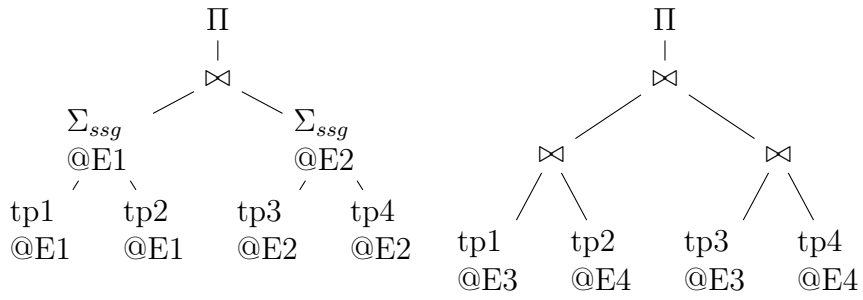
FedX join ordering heuristics do not take into account overlap with previously bounded variables to promote joins and and avoid Cartesian products between different subqueries.

For *Q* and *Federation2*, FedX builds the plan given in Figure 7.3b. Exclusive group composed of tp_1 and tp_3 is to be evaluated first, then exclusive group composed of tp_2 and tp_4. Both exclusive groups are Cartesian products, and the number of transferred tuples may be seriously increased by the use of these two exclusive groups. Populating *Federation2* with the same data as *Federation1*, the number of transferred tuples increases five orders of magnitude when exclusive groups are used (Table 7.3).

7.4.2 ANAPSID

ANAPSID [2] is an adaptive federated query engine that hides network delays during query execution. ANAPSID source selection depends on `SELECT` queries that provide the different predicates in the triples that each endpoint stores. These queries can be sent to the federation members once before the execution of a set of queries, and re-sent if any source update is suspected. For each triple pattern, the most probable sources to provide relevant data are determined using heuristics [60]. [5] If these heuristics lead to select more than one source, then the triple pattern is sent individually to the sources to retrieve data from. The other triple patterns, are grouped into *star-shaped groups*, i.e., triple patterns that share a variable, that are to be sent to the endpoints together in order to reduce the size of intermediate results, and the number of transferred from endpoints to the query engine. For query *Q* and *Federation1*, ANAPSID builds two star-shaped groups, one composed of tp_1 and tp_2 to be sent to *E1*, and one composed of tp_3 and tp_4 to be sent to *E2*. ANAPSID heuristic, that chooses the endpoints where other triple patterns that share the subject variable, makes the choice to send tp_3 only to *E2* (instead of *E1* and *E2*). In order to increase the parallelism, joins are com-

5. In practice, these heuristics have been shown to be accurate for most queries, but they could prune relevant sources needed to provide a complete answer.



(a) Federation1 (b) Federation2

Figure 7.4 – ANAPSID execution plans for query Q (Listing 7.1) and federations *Federation1* and *Federation2* (Table 7.1)

bined into bushy tree plans, and joins are sorted according to their estimated selectivity. Adaptive implementations of joins are used to hide source delays and produce answers incrementally. Current version [6] uses xgjoin and nested hash join implementations for symmetric joins and dependent joins respectively. The choice between both implementations depend on the estimated selectivity of their operands.

Table 7.4 – Positive impact of the use of star-shaped groups (SSG) on the number of transferred tuples (TT) in *Federation1* (Table 7.1) and Q (Listing 7.1)

With SSG		Without SSG	
Executed Subquery	# TT	Executed Subquery	# TT
tp_1 . tp_2	141	tp_1	1,700
tp_1 . tp_2 . tp_3 . tp_4	142	tp_1 . tp_2	1,841
		tp_1 . tp_2 . tp_3	178,299
		tp_1 . tp_2 . tp_3 . tp_4	191,317

For Q and *Federation1*, ANAPSID builds the plan given in Figure 7.4a. This plan is evaluated using a nested hash join, as an heuristic determines that tp_1 is selective enough to choose this physical operator. Using the same Virtuoso endpoints as in the previous section, we executed the query using ANAPSID [7], and the number of transferred tuples with and without star-shaped groups are presented in Table 7.4, we can observe a reduction of two orders of magnitude due to the star-shaped groups. The number of transferred tuples, by the execution without star-shaped groups, is clearly different from the one given in Table 7.2, for FedX without exclusive group as both engines use different implementations of join.

ANAPSID does not group triple patterns without variables in common in *star-shaped groups*, thus it avoids expensive evaluation of Cartesian products by the endpoints, and large amounts of

6. Version of May 14th, 2014
7. Version of May 14th, 2015

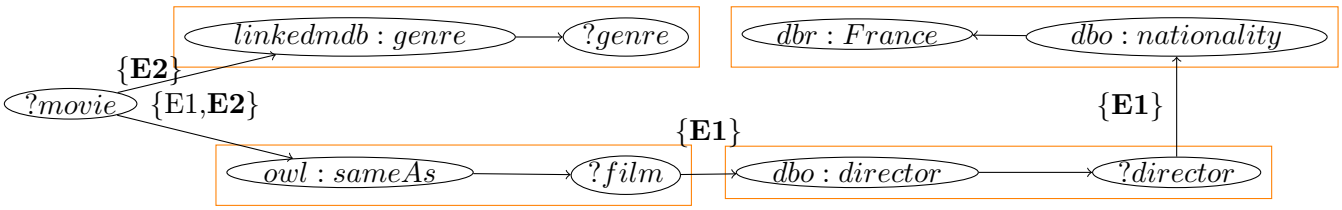


Figure 7.5 – HiBISCuS labelled hypergraph for query Q (Listing 7.1) and *Federation1* (Table 7.1). Sources selected by HiBISCuS appear in **bold**.

transferred data. For Q and *Federation2*, ANAPSID builds the plan given in Figure 7.4b, where no star-shaped groups were built. The left-most join is evaluated using a nested hash join, because an heuristic determines that tp_1 is selective enough to choose this physical operator, and the two other joins are evaluated using xgjoin implementation as the heuristic does not consider these operands as selective enough. The number of tuples transferred by this plan is the same as in Table 7.4 without star-shaped groups.

7.5 Source Selection Strategies for SPARQL endpoints

Recently, strategies to improve the source selection performed by federated query engines like FedX have been proposed. A first approach is to prune the sources that cannot contribute to a query answer [69], join-aware approaches, and a second approach is to prune sources that can only provide redundant data [38, 70], duplicate-aware approaches.

7.5.1 Join-Aware Source Selection Strategies

HiBISCuS [69] source selection approach has been proposed to reduce the number of selected sources. The reduction is achieved by annotating sources with the authority of their resource URIs. The authority of a URI is defined (in [69]) by the first two components of the URI [8]. For instance, in the URI *http://dbpedia.org/resource/Jean_Renoir*, the authority is *http://dbpedia.org*. HiBISCuS is based on the idea that when there is a join between two triple patterns, then these triple patterns should be only evaluated in sources that have authorities in common. In other words, its goal is to avoid empty joins by pruning sources that do not have authorities in common.

Basic graph patterns in queries are represented as directed labeled hypergraphs. Each query triple pattern is represented as a directed hyperedge, the hyperedge connects the vertex that represents the

8. Internet standard about URIs is available at http://tools.ietf.org/html/rfc3986

subject with an hypervertex that contains the vertexes that represent the predicate and the object. Hyperedges are labeled with the set of sources that should be contacted in order to retrieve data for the triple pattern. Joins are represented as multiple arcs incident in the same vertex. A join can produce a non empty answer if the incident arcs have sources with at least one common authority. Therefore, if a source has no common authority with the sources present in the other incident arcs, such source can be safely pruned from the hypergraph label, i.e., such source cannot contribute to produce any query answer.

Table 7.5 – HiBISCuS summaries for *Federation1* (Table 7.1)

(a) Endpoint *E1*

Predicate	Subject Authority	Object Authority
http://dbpedia.org/ontology/director	http://dbpedia.org	http://dbpedia.org
http://www.w3.org/2002/07/owl#sameAs	**http://dbpedia.org** **http://data.nytimes.com**	http://linkedgeodata.org http://dbpedia.org
http://dbpedia.org/ontology/nationality	http://dbpedia.org	http://dbpedia.org

(b) Endpoint *E2*

Predicate	Subject Authority	Object Authority
http://www.w3.org/2002/07/owl#sameAs	**http://data.linkedmdb.org**	http://dbpedia.org, http://mpii.de, http://sws.geonames.org, http://zitgist.com
http://data.linkedmdb.org/resource/movie/genre	**http://data.linkedmdb.org**	http://data.linkedmdb.org

For *Federation1* (Table 7.1) annotations are summarized in Table 7.5. These annotations can be used to determine the set of endpoints that can have data to evaluate a triple pattern. For example, the triples with predicate *owl:sameAs*, if they are available through the endpoint E1, then their subject URIs start with *http://dbpedia.org* or *http://data.nytimes.com*, while if they are available through endpoint E2, then their subject URIs start with *http://linkedgeodata.org* or *http://dbpedia.org*. Therefore, for triple pattern *<http://data.nytimes.com/474522189480777706853> owl:sameAs ?o*, there is not doubt that E2 does not have triples that match this triple pattern.

Consider query *Q* (Listing 7.1) and *Federation1* (Table 7.1). Its directed labeled hypergraph representation is shown in Figure 7.5. The initial labeling of the hyperedges is straightforward, first and second triple patterns can only be evaluated by E1, the fourth triple pattern can only be evaluated by E2, and the third triple pattern can be evaluated by both E1 and E2. Then for each vertex with multiple incident arcs of the same type (in or out), the authority annotations of the sources present in the labels of the arcs are intersected to obtain the common authorities. Sources with no common authorities are pruned from the labels. In the example, the vertex *?movie* has two outgoing arcs, the authorities associated to these arcs are { *http://data.linkedmdb.org* } and { *http://data.linkedmdb.org*, *http://dbpedia.org*, *http://data.nytimes.com*}, and their intersection is {

http://data.linkedmdb.org }. Endpoint *E1* is pruned from the label of the outgoing arc of vertex *?movie* because its set of authorities { *http://dbpedia.org*, *http://data.nytimes.com*} does not include *http://data.linkedmdb.org*.

7.5.2 Duplicate-Aware Source Selection Strategies

Recently, BBQ [38] and DAW [70] propose duplicate-aware strategies for selecting sources for federated query engines. Both approaches use sketches to estimate the overlapping among sources. Benefit-Based Query routing (BBQ) extends ASK queries with Bloom filters [15] that provide a summary of the results, in order to prune sources that provide low benefits. DAW uses a combination of Min-Wise Independent Permutations (MIPs) [17], and triple selectivity information to estimate the overlap between the results of different sources. Based on how many new query results are expected to be found, sources that are below predefined benefits, are discarded and not selected.

For *Federation1* (Table 7.1) and tp_3 of query Q (Listing 7.1), both DAW and BBQ select both sources as they have no triple patterns in common for *owl:sameAs*, then only selecting both it is possible to be sure that all the answers will be produced.

DAW duplicate-aware source selection strategy is detailed in the following section.

7.5.3 DAW

DAW [70] duplicate-aware source selection strategy has as input the set of sources that can provide data for each of the query triple patterns, and as output the set of sources that should be contacted by the federated query engine.

DAW is comprised of two parts, in its first part, it ranks the capable sources according to how much new data they can provide. Then, in the second part the sources that provide less than a predefined among of data are pruned.

Table 7.6 – *Federation3* endpoints that have triples with predicates in the query Q (Listing 7.1)

Q triple pattern		Federation3
tp_1	?director dbo:nationality dbr:France	E1, E3
tp_2	?film dbo:director ?director	E1, E4
tp_3	?movie owl:sameAs ?film	E1, E2, E3
tp_4	?movie linkedmdb:genre ?genre	E2, E4

Table 7.7 – DAW's input and output for query Q (Listing 7.1) and *Federation3* (Table 7.6)

Triple Pattern	input	output
tp_1	{ E1, E3 }	{ E1 }
tp_2	{ E1, E4 }	{ E1 }
tp_3	{ E1, E2 , E3 }	{ E1, E3 }
tp_4	{ E2, E4 }	{ E2 }

Consider the query Q (Listing 7.1), *Federation3* (Table 7.6) comprised of endpoints *E1-E4* from *Federation1* and *Federation2* (Table 7.1), and endpoints populated with data from DBpedia [9] and LinkedMDB [10], and a threshold of zero, i.e., only sources estimated to return no new data are pruned. Endpoints *E1* and *E3* have all the triples of DBpedia with predicate *dbo:nationality*, endpoints *E1* and *E4* have all the triples of DBpedia with predicate *dbo:director*, endpoints *E2* and *E4* have all the triples of LinkedMDB with predicate *linkedmdb:genre*, endpoints *E1* and *E3* have all the triples of DBpedia with predicate *owl:sameAs*, and endpoints *E2* and *E3* have all the triples of LinkedMDB with predicate *owl:sameAs*.

Table 7.7 shows the input and output of DAW [11] for query Q (Listing 7.1), and *Federation3* (Table 7.6). Because both *E1* and *E3* have all the triple patterns with predicate *dbo:nationality* available in the federation, then any of them may be ranked first by DAW's first part. And consequently, the one that has been ranked second has been pruned by the second part of DAW, because having exactly the same triples than the first, it does not provide any new data. For tp_3, only $E3$ can be ranked first, and it is the only one that has all the triples with predicate *owl:sameAs* available in the federation. Even if *E1* does not provide any new data for tp_3, the overlapping detection used by DAW fails to assess this fact, and it also selects *E1* for tp_3.

DAW uses Min-Wise Independent Permutations (MIPs) [17] to compute summaries of the data accessible through the endpoints, and use these summaries to approximate the overlap among triples accessible through different endpoints. The set of triples accessible through an endpoint with a given predicate is represented using a vector of integer identifiers, with each identifier being the hash code of a triple subject and object string representation concatenated. From this vector, k random permutations are generated. Each random permutation is generated using a linear hash function of the form: $h_i(x) := (a_i * x + b_i) \, mod \, U$, with U a big prime number, and a_i, b_i fixed random numbers per function. Finally, the minimum element of each permutation is included in the MIP vector that

9. DBpedia3.9 subset as in FedBench [73]
10. Version from January 19th, 2010
11. We implemented DAW ourselves because its authors cannot provide its code

summarizes the set of triples. The principle of MIPs is that each set element has the same probability of becoming the minimum in a given random permutation. Thus, the resemblance of two sets, S_1 and S_2, can be approximated using their two MIP vectors, V_1 and V_2, as the number of positions where the vectors have the same value divided by the number of permutations [12]. The union of two MIP vectors can be computed by taking the minimum element in each position. And the approximate overlap of two sets can be computed as:

$$Overlap(S_1, S_2) \approx \frac{Resemblance(V_1, V_2) \times (|S_1| + |S_2|)}{Resemblance(V_1, V_2) + 1}$$

Listing 7.3 – Triples accessible through E2

```
<http://dbpedia.org/resource/Wymore,_Nebraska> <http://www.w3.org/2002/07/owl#sameAs>
    <http://linkedgeodata.org/triplify/node151438039> .
<http://dbpedia.org/resource/Carrara> <http://www.w3.org/2002/07/owl#sameAs>
    <http://linkedgeodata.org/triplify/node61753614> .
<http://data.nytimes.com/N48490752132683526173> <http://www.w3.org/2002/07/owl#sameAs>
    <http://dbpedia.org/resource/Maggie_Gyllenhaal> .
<http://dbpedia.org/resource/Gimingham> <http://www.w3.org/2002/07/owl#sameAs>
    <http://linkedgeodata.org/triplify/node29829116> .
<http://dbpedia.org/resource/Enborne_Row> <http://www.w3.org/2002/07/owl#sameAs>
    <http://linkedgeodata.org/triplify/node309083295> .
```

Listing 7.4 – Triples accessible through E1

```
<http://data.linkedmdb.org/resource/film/34726> <http://www.w3.org/2002/07/owl#sameAs>
    <http://dbpedia.org/resource/The_Conversation> .
<http://data.linkedmdb.org/resource/film/22058> <http://www.w3.org/2002/07/owl#sameAs>
    <http://dbpedia.org/resource/The_Bad_Lands> .
```

To illustrate how DAW indexes are computed and used to state overlapping among set of triples, suppose that E1 has only the five triples given in Listing 7.3, E2 has only the two triples given in Listing 7.4, and E3 has only the seven triples Listings 7.3 and 7.4.

MIP vectors for the five triples in Listing 7.3, and seven triples in Listings 7.3 and 7.4 are shown in Tables 7.9 and 7.10. These tables also show the random permutations of the set of integer identifiers that represent the set of triples, and Table 7.8 present the values of a_i and b_i used to generate the permutations. These two MIP vectors have three common elements, and they are highlighted in **bold**. Therefore, their resemblance is $\frac{3}{5}$, and the overlap between their set of triples is approximated

12. If the vectors where computed using different number of permutations, then their minimum is used, at the price of a precision loss [70]

Table 7.8 – Values of a_i and b_i used to compute the random permutations $h_i(x) := (a_i * x + b_i) \, mod \, U$, the value of U is set to 991205981

i	a_i	b_i
1	-1286082570	-1558221506
2	-1343278692	1858951374
3	-390706926	1793532194
4	-488269753	950550283
5	1355994690	-2014772492
6	-1870576674	-1059013661
7	893439232	-588725372

Table 7.9 – Random permutations $h_i(x) := (a_i * x + b_i) \, mod \, U$, using values given in Table 7.8 for the triple set given in Listing 7.3

set	h_1	h_2	h_3	h_4	h_5
-811800833	841080043	-206443726	320345616	-243051999	697245781
275212389	172427596	-181672521	-826325700	-222313109	731627681
-712626126	888787181	761638633	176958537	589476492	-372294696
247982479	-690004568	-134332500	439964720	800189876	-149725649
-902440067	960825948	406875037	370186895	-121940298	-875555026
MIP vector	**-690004568**	-206443726	**-826325700**	-243051999	**-875555026**

as 4.5.

In the federation of our example, with the endpoints populated with the triples from DBpedia and LinkedMDB, computing the DAW index takes 109.75 secs, and such index weights 837K. A first limitation of DAW is the index computation; if the user that wants to execute the query has to compute it, then she should have access to all the federation triples, in which case she could do better than computing these summaries to choose where to execute each triple pattern; if the data publisher is to provide the summaries, the summaries are to be transferred and be kept up to date in order to avoid stale data, and the selection the wrong set of sources. A second limitation of DAW is the accuracy of overlap detection. In order to produce good quality overlap assessment the endpoints should have similar number of triples per predicate, and this restriction is quite strong. A third limitation of DAW is the source selection time. Computing set overlap, vector resemblance and union, are operations that take a non-negligible amount of time. In any case, the computation and transfer of indexes to compute approximate overlapping, is too expensive in the context of data replication, where more concise descriptions of the replicated fragments can be used to produce a better quality overlap assessment.

Table 7.10 – Random permutations $h_i(x) := (a_i * x + b_i) \, mod \, U$, using values given in Table 7.8 for the triple set comprised of triples given in Listings 7.3 and 7.4

set	h_1	h_2	h_3	h_4	h_5	h_6	h_7
1686545447	-33385544	614022453	-320353568	356864895	-673845502	-767538507	-887357820
-1075019727	-355732047	-734799958	-454287980	-852368222	775751062	52502276	901473831
-811800833	841080043	-206443726	320345616	-243051999	697245781	-319200158	212006532
275212389	172427596	-181672521	-826325700	-222313109	731627681	984619036	-388612383
-712626126	888787181	761638633	176958537	589476492	-372294696	939066082	-216700959
247982479	-690004568	-134332500	439964720	800189876	-149725649	91009765	-240105852
-902440067	960825948	406875037	370186895	-121940298	-875555026	503564617	-117410079
MIP vector	**-690004568**	-734799958	**-826325700**	-852368222	**-875555026**	-767538507	-887357820

7.6 Strategies to overcome availability limitations in Linked Data

Public SPARQL endpoints are typically provided by organizations with limited amount of resources available, and their intensive use for query processing compromises their reliability, availability, and performance.

Linked Data fragments approach (LDF) [80, 81] proposes to improve Linked Data availability by moving query execution load from servers to clients. A client is able to execute locally a restricted SPARQL query by downloading fragments required to execute the query from an LDF server through a simple HTTP request.

Moreover, some optimizations and a local data store to improve LDF client performance have also been proposed [**?**], these optimizations allow clients to cache fragments locally and decreases the load on the LDF server. LDF chooses a clear tradeoff by shifting query processing to clients, at the cost of slower query execution.

7.7 Summary

In distributed databases, data fragmentation and replication improve data availability and query performance [62]. Data fragmentation is tailored for representative queries; fragments are smartly allocated and replicated across servers for balancing workload and reducing size of intermediate results. Linked Data [13] is intrinsically a federation of autonomous participants where federated queries are unknown to a single participant, and a tight coordination of data providers is difficult to achieve. Consequently, federated query engines cannot rely on properties ensured by an allocation algorithm, and new strategies to perform source selection are needed.

8

Fedra

8.1 Motivations

Existing SPARQL federated query engines do not support replicated data. To illustrate, we replicated the DBpedia dataset [1] and defined two federations. The first is composed of one mirror of DBpedia, and the second of two identical mirrors of DBpedia. We used FedX [74] (Section 7.4.1) and ANAPSID [2] (Section 7.4.2) to execute the query in Figure 8.1a against both federations. In the first federation, both engines produced all the query answers in less than 5 seconds.

On the other hand, for the second federation, the query engines, having no knowledge about the relationships among the mirrors of DBpedia, contact both data sources. In this way, performance in terms of execution time and number of transferred tuples, is seriously degraded as depicted in Figure 8.1b. For both engines the execution time and number of transferred tuples increase more than 250 times when a second replica of DBpedia is added to the federation. Having no knowledge about the relationships among the mirrors of DBpedia, both query engines have to retrieve twice all the triples that match each of the triple patterns of the query, instead of evaluating the joins in the endpoints and retrieving only the query answers. For the first triple pattern, the number of triples is greater than 4 millions. This number is likely to be higher than the maximum number of result

1. DBpedia2015, http://wiki.dbpedia.org/Downloads2015-04

```
select distinct ?p ?m ?n ?d where {
  ?p  dbprop:name ?m .
  ?p  dbprop:nationality ?n .
  ?p  dbprop:doctoralAdvisor ?d
}
```

(a) DBpedia Query

#DBpedia Replicas	FedX		ANAPSID	
	ET (s)	NTT	ET (s)	NTT
1	4.80	8,230	2.61	8,229
2	2,678.10	2,260,006	3,415.24	8,337,702

(b) Query Execution

Figure 8.1 – DBpedia query and its Execution Time (ET) and Number of Transferred Tuples (NTT) during query execution against federations with one and two replicas of DBpedia

rows that the endpoint is allowed to send, in consequence it risks to produce incomplete answers.

Furthermore, if the DAW [70] or BBQ [38] approaches (Section 7.5.2) were used, data providers and consumers resources would be used to compute and download data summaries. These approaches could select different DBpedia endpoints per triple pattern, and execute the join between retrieved data at the federated engine level.

Of course, if federated query engines would know that both endpoints are mirrors of DBpedia, the source selection pruning could be done more efficiently, i.e., only one source would be selected to execute the query. This problem is even more challenging if we consider that one endpoint can partially replicate data from several RDF datasets, i.e., only fragments of several datasets are replicated, e.g., to speed up query execution of some queries.

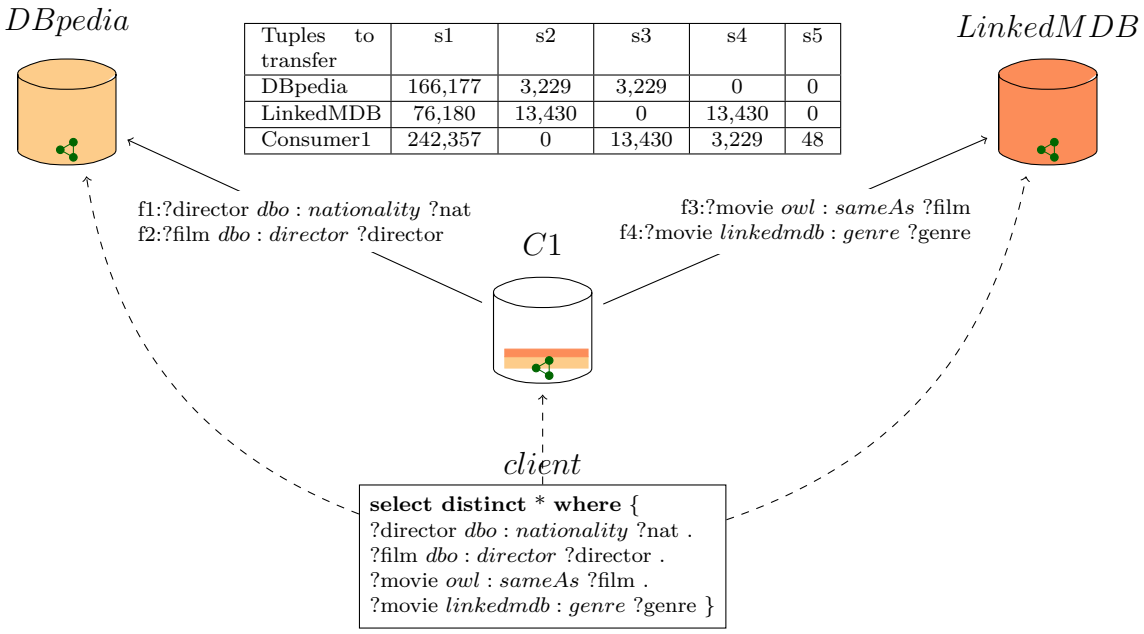


Figure 8.2 – Client defines a federation composed of DBpedia, LinkedMDB, and C1 endpoints with four replicated fragments

Suppose a Web application poses federated queries against endpoints DBpedia and LinkedMDB. In order to speed up the queries, a data consumer endpoint $C1$ with replicated fragments has been

installed as in Figure 8.2. Fragments are defined as simple `CONSTRUCT` SPARQL queries with one triple pattern. Fragments allow for the re-organization of RDF data *on* $C1$ to better address needs of data consumers.

Even in this simple setup, processing our running query against a federation including DBpedia, LinkedMDB, and $C1$ raises the problem of source selection with fragment replication (SSP-FR). There are at least five options to select sources for executing this query; these choices produce different number of transferred tuples as shown in Figure 8.2:

(i) If no information about replicated fragments is available, all sources may be selected to retrieve data for all the triple patterns. The number of transferred tuples is given in the solution $s1$. This will be the behavior of a federated query engine like FedX that ensures answer completeness. [2]

(ii) Endpoints DBpedia and LinkedMDB could be chosen, in this case the number of transferred tuples is given in $s2$. The number of transferred tuples in $s2$ is less than $s1$ since some joins could be executed at DBpedia and LinkedMDB.

(iii) Another choice may be to use the $C1$ endpoint in combination with either DBpedia or LinkedMDB ($s3, s4$). This produces the same number of transferred tuples as in $s2$, but they have the advantage of accessing less public endpoints.

(iv) A last choice could be to use the $C1$ endpoint to retrieve data for all the triple patterns ($s5$). This solution profits from replicated fragments to execute opportunistic joins at $C1$; thus, it is able to achieve the *best* performance in terms of the number of transferred tuples.

As the number of transferred tuples increases, the availability of the contacted SPARQL endpoints can be affected. A replication aware federated query engine could select the best sources to reduce the number of transferred tuples while preserving answer completeness. In this thesis, we formally address the following problem: Given a SPARQL query and a set of relevant SPARQL endpoints with replicated fragments, choose the SPARQL endpoints to contact in order to produce a complete query answer and transfer the minimum amount of data. We aim to develop an algorithm that produces solution $s5$ whenever possible, providing as output the sources to be used by a federated query engine.

2. In order to preserve joins between different endpoints, each triple pattern should be posed to each endpoint individually.

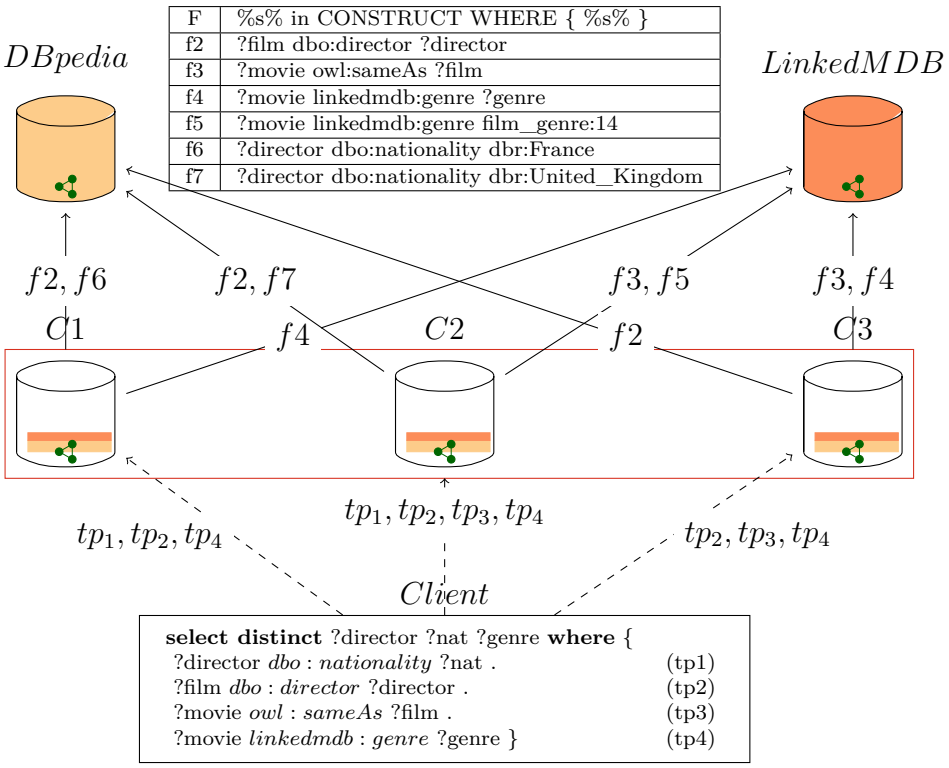


Figure 8.3 – Client defines a federation composed of $C1$,$C2$, and $C3$ that replicates fragments $f2-f7$

8.2 Definitions and Problem Description

This section introduces definitions and the source selection problem with fragment replication (SSP-FR).

8.2.1 Definitions

Fragments are set used to replicate RDF data. The data of a fragment is defined by means of the dataset public endpoint, or *authoritative endpoint*, and a `CONSTRUCT` query with *one* triple pattern.

Definition 14 (Fragment). *A fragment is a tuple* $f = \langle u, s \rangle$

- *u is the non-null URI of the authoritative endpoint where* f *is available;*
- *s is a CONSTRUCT query with one triple pattern.*

Without loss of generality, s is limited to one triple pattern as in [39], [80]; this reduces the complexity of *fragment containment* problem as described in Definition 15. Additionally, we assume replicated fragments comprise RDF data accessible from public endpoints, i.e., the authoritative endpoints of the replicated fragments are disjoint with data consumer endpoints. This will allow data consumers to re-organize RDF data replicated from different public endpoints to fit in this way,

their needs and requirements.

In this work, we make the following assumptions:

(i) Fragments are replicated from public endpoints, and there is just one level of replication.

(ii) Fragments are read-only and perfectly synchronized; the fragment synchronization problem is studied in [39], while querying fragments with divergence may be addressed as in [57].

(iii) For the sake of simplicity, we suppose that RDF data accessible through the endpoints are described as fragments.

To illustrate, consider the federation given in Figure 8.3. This federation extends the setup in Figure 8.2. Suppose three Web applications pose queries against DBpedia and LinkedMDB. To speed up query processing, data consumer endpoints: $C1$, $C2$, and $C3$ with replicated fragments have been configured.

At startup, the federated query engine loads the fragment descriptions for each of the federation endpoints, and computes both the *fragment* and *containment* mappings. The **fragment mappings** is a function that maps fragments to a set of endpoints; the **containment mapping** is based on *containment* relation ($f_l \sqsubseteq f_k$) described in the Definition 15.

Two fragments loaded from two different endpoints, that have the same authoritative endpoint and equivalent `CONSTRUCT` queries, are concatenated in the fragment mapping. For example, the federated engine loads fragments $\langle$*http://dbpedia.org/sparql, ?film db:director ?director*$\rangle$ from $C1, C2, C3$, computes equivalence, and adds in its fragment mapping $\langle$*http://dbpedia.org/sparql, ?film db:director ?director*$\rangle \rightarrow$ *{C1,C2,C3}*.

We adapt definitions of containment and equivalence [26], [34] for the case of triple pattern queries.

Definition 15 (Triple Pattern Containment and Equivalence)**.** *Let $TP(D)$ denote the result of execution of the triple pattern TP against an RDF dataset D. Let TP_1 and TP_2 be two triple patterns. We say that TP_1 is contained in TP_2, denoted by $TP_1 \sqsubseteq TP_2$, if for any RDF dataset D, $TP_1(D) \subseteq TP_2(D)$. We say that TP_1 is equivalent to TP_2, denoted by $TP_1 \equiv TP_2$, if $TP_1 \sqsubseteq TP_2$ and $TP_2 \sqsubseteq TP_1$.*

As stated in Theorem 4.2.1 [26], containment testing can be achieved using a containment mapping (Definition 6 [26]). It amounts to finding a substitution of the variables in the triple patterns. [3]

3. The substitution operator preserves URIs and literals, only variables are substituted.

$TP_1 \sqsubseteq TP_2$, iff there is a substitution θ such that applying θ to TP_2 returns the triple pattern TP_1. Solving the decision problem of triple pattern containment between TP_1 and TP_2, $TP_1 \sqsubseteq TP_2$, requires to check if $TP1$ imposes at least the same restrictions as TP_2 on the subject, predicate, and object positions, i.e., TP_1 should have at most the same number of unbounded variables as TP_2. Hence, testing triple pattern containment has a complexity of $O(1)$.

For the federation in Figure 8.3, $f5 \sqsubseteq f4$ because $f4$ and $f5$ share the same authoritative endpoint and there is a substitution θ defined as $\theta(?genre) = film\_genre : 14$, $\theta(?movie) = ?movie$, and applying θ to $f4$ returns $f5$. After identifying a substitution θ for all pair-wise fragments, it is straightforward to compute a **containment mapping** for a federation of SPARQL endpoints.

We can rely on fragment descriptions and the containment property to determine relevant fragments to a query. Relevant fragments contain relevant RDF data to each of the triple patterns of the query. A fragment is *relevant* to a query Q, if it is relevant to *at least one* triple pattern of the query.

We adapt Definitions 11 and 12 to triple pattern fragments in the following definition:

Definition 16 (Fragment relevance)**.** *Let f be a fragment defined by a triple pattern TP_1. Let TP_2 be a triple pattern of a query Q. f is relevant to Q if $TP_2 \sqsubseteq TP_1$ or $TP_1 \sqsubseteq TP_2$.*

Table 8.1a shows the relevant fragments to the triple patterns in query Q, and the endpoints that provide these fragments. For example, the triple pattern $tp1$ has two relevant fragments: $f6$ and $f7$, and triple pattern $tp4$ has two relevant fragments: $f4$ and $f5$. Fragment $f4$ can produce the complete answer of $tp4$ because $f5 \sqsubseteq f4$, while both $f6$ and $f7$ are required to answer $tp1$. Even if none of $f6$ nor $f7$ contains $tp1$, and some other triple patterns in the DBpedia endpoint may have the same predicate as $tp1$, in the given federation composed by $C1$, $C2$, and $C3$, where no other fragment can provide data for $tp1$, retrieving data from both $f6$ and $f7$ leads to a complete answer wrt $tp1$ and the given federation.

8.2.2 Source Selection Problem with Fragment Replication (SSP-FR)

Given a SPARQL query Q, a set of SPARQL endpoints E, the set of fragments F that have been replicated by at least one endpoint in E, a fragment mapping *endpoints()*, a containment mapping $\sqsubseteq$. The Source Selection Problem with Fragment Replication (SSP-FR) is to assign to each triple

Table 8.1 – Q Relevant fragments, and source selections that lead to produce all the obtainable answers for the federation given in Figure 8.3

(a) Relevant Fragments

Q triple pattern		RF	Endpoints
tp_1	?director dbo:nationality ?nat	f6	C1
		f7	C2
tp_2	?film dbo:director ?director	f2	C1,C2,C3
tp_3	?movie owl:sameAs ?film	f3	C2,C3
tp_4	?movie linkedmdb:genre ?genre	f4	C1,C3
		f5	C2

(b) Source selections

TP	D_0(tp)	D_1(tp)	D_2(tp)
tp_1	{C1,C2}	{C1,C2}	{C1,C2}
tp_2	{C1,C2,C3}	{C1}	{C3}
tp_3	{C2,C3}	{C2}	{C3}
tp_4	{C1,C2,C3}	{C3}	{C3}
Tuples to transfer	421,675	170,078	8,953

pattern in Q, the set of endpoints from E that need to be contacted to answer Q. A solution of SSP-FR corresponds to a mapping D that satisfies the following properties:

1. **Answer completeness:** sources selected in D lead engines to produce complete query answers.
2. **Data redundancy minimization:** *cardinality(D(tp))* is minimized for all triple pattern *tp* in Q, i.e., redundant data is minimized.
3. **Data transfer minimization:** executing the query using the sources selected in D minimizes the transferred data.

We illustrate SSP-FR on running query Q of Figure 8.3. Table 8.1a presents relevant fragments for each triple pattern. Table 8.1b shows three $D(tp)$ that ensure the answer completeness property. It may seem counterintuitive that these three *D(tp)* do ensure the answer completeness property, as they do not include existing DBpedia triples for *dbo:nationality* predicate with object different from *dbr:France* and *dbr:United_Kingdom*, but as they are not included in endpoints in E, these triples are inaccessible to the federation. Even if D_1 and D_2 minimize the number of selected endpoints per triple pattern, only D_2 minimizes the transferred data. Indeed, executing $tp1, tp2, tp3$ against replicated fragments that are located in the same data consumer endpoint will greatly reduce the number of transferred tuples.

The DAW [70] and BBQ [38] approaches (Section 7.5.2) are not designed for solving SSP-FR. Indeed, they do not take into account replicated data, and may produce a solution as D_1. The FEDRA algorithm exploits properties of the replicated fragments and is able to find solution D_2.

8.3 Fedra: an Algorithm for SSP-FR

The goal of FEDRA is to reduce data transfer by taking advantage of the replication of relevant fragments for several triple patterns on the same endpoint. Algorithm 5 proceeds in four main steps:

Algorithm 5 FEDRA Source Selection algorithm

Require: Q: SPARQL Query; F: set of Fragment; endpoints : Fragment → set of Endpoint; $\sqsubseteq$: TriplePattern × TriplePattern → boolean
Ensure: selectedEndpoints: TriplePattern → set of Endpoint.

```
1: function SOURCESELECTION(Q,F,endpoints,⊑)
2:     triplePatterns ← get triple patterns in Q
3:     R, E ← ∅, ∅
4:     for all tp ∈ triplePatterns do
5:         R(tp) ← RELEVANTFRAGMENTS(tp, F)                ▷ Relevant fragments as in Definition 16
6:         R(tp) ← {{f : f ∈ R(tp) : tp ⊑ f}} ⋃ {{f} : f ∈ R(tp) : f ⊑ tp ∧ ¬(∃g : g ∈ R(tp) : f ⊏ g ⊑ tp)}
7:         E(tp) ← { (⋃ endpoints(f) : f ∈ fs) : fs ∈ R(tp) }
8:     end for
9:     bgps ← get basic graph patterns in Q
10:    for all bgp ∈ bgps do
11:        UNIONREDUCTION(bgp, E)                          ▷ endpoints reduction for multiple fragment triples
12:        BGPREDUCTION(bgp, E)                            ▷ endpoints reduction for the bgp triples
13:    end for
14:    for all (tp, E(tp)) ∈ E do
15:        selectedEndpoints(tp) ← for each set in E(tp) include one element
16:    end for
17:    return selectedEndpoints
18: end function
```

Algorithm 6 Union reduction algorithm

Require: tps : set of TriplePattern; E : TriplePattern → set of set of Endpoint

```
19: procedure UNIONREDUCTION(tps, E)
20:     triplesWithMultipleFragments ← { tp : tp ∈ tps ∧ cardinality(E(tp)) > 1 }
21:     for all tp ∈ triplesWithMultipleFragments do
22:         commonSources ← (⋂ f : f ∈ E(tp))               ▷ get sources in all the subsets in E(tp)
23:         if commonSources ≠ ∅ then
24:             E(tp) ← { commonSources }
25:         end if
26:     end for
27: end procedure
```

Algorithm 7 Basic graph pattern reduction algorithm

Require: tps : set of TriplePattern; E : TriplePattern → set of set of Endpoint

```
28: procedure BGPREDUCTION(tps, E)
29:     triplesWithOneFragment ← { tp : tp ∈ tps ∧ cardinality(E(tp)) = 1 }
30:     (S, C) ← minimal set covering instance using triplesWithOneFragment◁E
31:     C' ← MINIMALSETCOVERING(S, C)
32:     selected ← get endpoints encoded by C'
33:     for all tp ∈ triplesWithOneFragment do
34:         E(tp) ← E(tp) ⋂ selected
35:     end for
36: end procedure
```

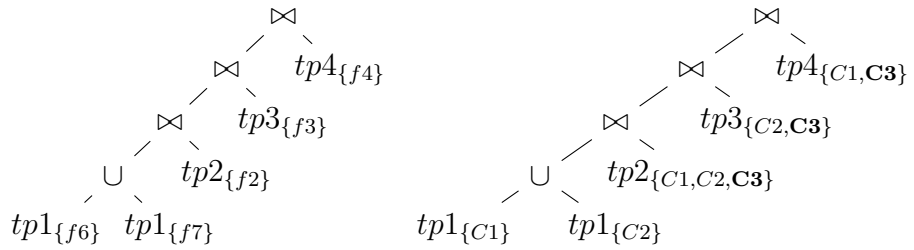
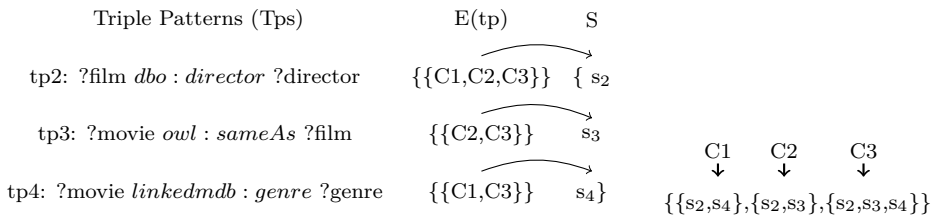


Figure 8.4 – Execution plan encoded in data structures R (left) and E (right); multiple subsets represent union of different fragments (ex. {f6}, {f7}); elements of the subset represent alternative location of fragments (ex. {C1,C3}); **bold** sources are the selected sources after set covering is used to reduce number of selected sources



(a) S instances (b) C instance

Figure 8.5 – S and C instances obtained from the set covering reduction (used by Algorithm 7) for the query Q and federation given in Figure 8.3

I. Identify relevant fragments for triple patterns, a Basic Graph Pattern (BGP) triple pattern can be contained in one fragment or a union of fragments (lines 5-6).

II. Localize relevant replicated fragments on the endpoints, e.g., Figure 8.4 (line 7).

III. Prune endpoints for the unions (line 11).

IV. Prune endpoints for the BGPs using a set covering heuristic (line 12).

Next, we illustrate how Algorithm 5 works on our running query Q and data consumer endpoints $C1, C2, C3$ from Figure 8.3. [4]

First, for each triple pattern, FEDRA computes relevant fragments in *R(tp)*, and groups them if they provide the same relevant data. For tp1, $R(tp1) \rightarrow \{\{f6\},\{f7\}\}$. For tp4, as $f5 \sqsubseteq f4$, $f5$ is safely removed at line 6, and $R(tp4) \rightarrow \{\{f4\}\}$. Second, FEDRA localizes fragments on endpoints in $E(tp)$. For tp1, $E(tp1) \rightarrow \{\{C1\},\{C2\}\}$. For tp4, $E(tp4) \rightarrow \{\{C1, C3\}\}$. Figure 8.4 shows the execution plans encoded in *R(tp)* and *E(tp)*. Triple patterns like tp1, with more than one relevant fragment, represent unions in the execution plan.

Procedure UNIONREDUCTION (cf. Algorithm 6) prunes non common endpoints, if possible, to access triple patterns from as few endpoints as possible. In our running example, it is not possible because there is no common endpoint that replicates both $f6$ and $f7$. However, if, for example, $f7$ were also replicated at $C1$, then only $C1$ would be selected to execute tp1.

Procedure BGPREDUCTION (cf. Algorithm 7) transforms the join part of $E(tp)$ (cf. Figure 8.4) into a set covering problem (cf. line 30). Each triple pattern is an element of the set to cover, e.g., tp2, tp3, tp4 correspond to s2, s3, s4 (cf. Figure 8.5a). And for each endpoint in $E(tp)$, we include the subset of triple patterns associated with that endpoint, e.g., for endpoint C1 we include the subset {s2,s4} as relevant fragments tp2 and tp4 are replicated by C1 (cf. Figure 8.5b). Line 31

4. As DBpedia is not included in the federation for processing Q, only fragments $f6$ and $f7$ are available to retrieve data for tp_1 and the engine will not produce all the answers that would be produced using DBpedia.

relies on an existing heuristic [41] to find the minimum set covering. In our example, it computes *C'={{s2,s3,s4}}*. Line 32 computes the selected endpoints, in our example, selected={ C3 }.

Finally, (Algorithm 5, line 15) chooses among endpoints that provide the same fragment and reduces data redundancy. For query Q, the whole algorithm returns D_2 of Table 8.1b.

Proposition 5. *Algorithm 5 has a time complexity of* $O(n.m^2)$*, with* n *the number of triple patterns in the query,* m *the number of fragments,* k *the number of endpoints,* l *the number of basic graph patterns in the query, and* $m \gg k \wedge k \gg l$ *holds.*

The upper bound given in Proposition 5 is unlikely to be reached, as it requires for all fragments to be relevant for each of the triple patterns. In practice (e.g., experiments from Section 8.4), even for high number of fragments (> 450), the source selection time remains low (Appendix B, section B.1).

Theorem 1. *If all the RDF data accessible through the endpoints of a federation are described as replicated fragments,* Fedra *source selection leads query engine to produce complete answers wrt the federation data.*

Proof. We assume that all the RDF data accessible through the endpoints are actually described as replicated fragments. By contradiction, we suppose that for a query Q, Fedra source selection leads the query engine to produce incomplete answers wrt the federation data, then there is at least one answer a that is a sound answer to Q using the federation data, that cannot be produced by the query engine using the sources selected by Fedra. Because it is Fedra source selection that prevents the query engine to produce a, then Fedra should have failed to include a source as relevant source for a query triple pattern. Without losing generality, suppose it is source s that Fedra has not included as relevant for triple pattern tp. But Fedra only prunes sources that provide redundant data for the triple patterns, if Fedra pruned s then its data is redundant, and if it is redundant the the sources selected by Fedra do lead to produce answer a. □

8.4 Experimental Study

The goal of the experimental study is to evaluate the effectiveness of Fedra. We compare the performance of federated SPARQL queries using FedX, DAW+FedX, Fedra+FedX, ANAPSID, DAW+ANAPSID, and Fedra+ANAPSID.

Table 8.2 – Dataset characteristics: version, number of different triples (#DT) and predicates (#P)

Dataset	Version date	#DT	#P
Diseasome	19/10/2012	72,445	19
Semantic Web Dog Food	08/11/2012	198,797	147
DBpedia Geo-coordinates	06/2012	1,900,004	4
LinkedMDB	18/05/2009	3,579,610	148
WatDiv1	_	104,532	86
WatDiv100	_	10,934,518	86

We expect to see that FEDRA selects less sources than the engines and DAW, and transfers less data from endpoints to the query engines.

Datasets: We use the real datasets: Diseasome, Semantic Web Dog Food, LinkedMDB, and DBpedia Geo-coordinates. Further, we consider two instances of the Waterloo SPARQL Diversity Test Suite (WatDiv) synthetic dataset [3, 4] with 10^5 and 10^7 triples. Table 8.2 shows the characteristics of these datasets.

Queries: We generate 50,000 queries (500 templates) for the WatDiv federation. We remove the queries that caused engines to abort execution, and queries that returned zero results. For the queries that return zero results, any source selection is good, and we are interested in queries where non trivial source selections are required. For the real datasets, we generate more than 10,000 queries using PATH and STAR shaped templates with two to eight triple patterns, that are instantiated with random values from the datasets. We include the DISTINCT modifier in all the queries, in order to make them susceptible to a reduction in the set of selected sources without changing the query answer.

Federations and random replication of fragments: For each dataset, we setup a ten consumer SPARQL endpoint federation (ten as in [70]). In order to produce federations where several opportunities to execute joins in the endpoints, and challenge FEDRA to find them, fragments were randomly replicated, and random queries were used to achieve it. For each federation endpoint, 100 random queries were selected. Each query triple pattern is executed as a SPARQL construct query with an LDF client [5] against an LDF server that exposes the whole dataset. The results are stored locally if not present in at least three consumer endpoints and a fragment definition is created. This replication factor of three was set to avoid federations with trivial solutions for FEDRA where all the fragments were replicated by one endpoint. In order to measure the number of transferred tuples, the federated query engine accesses data consumer endpoints through a proxy.

5. https://github.com/LinkedDataFragments, March 2015.

Statistical tests: The Wilcoxon signed rank test [86] for non-uniform paired data is used to study the statistical significance of the obtained results.

Implementations: FedX 3.1 [6] and ANAPSID [7] have been modified to replace their source selection strategies by FEDRA and DAW [70]. Thus, each engine can use the selected sources to perform its own optimization strategies. FEDRA and DAW [8] are implemented in both Java 1.7 and Python 2.7.3. Thus, FEDRA and DAW are integrated in FedX (Java) and ANAPSID (Python), reducing the performance impact of including these new source selection strategies. Proxies are implemented in Java 1.7. using the Apache HttpComponents Client library 4.3.5 [9]. We used R [10] to compute the Wilcoxon signed rank test [86].

Hardware and configuration details: The Grid'5000 testbed [11] is used to run the experiments. In total 11 machines Intel Xeon E5520 2.27 GHz, with 24GB of RAM in the grenoble site are used for each execution. Ten machines are used to host the consumer SPARQL endpoints, and one to run the federated query engines. Federated query engines use up to 7GB of RAM. Consumer SPARQL endpoints are deployed using Virtuoso 7.2.1 [12]. Virtuoso parameters *number of buffers* and *maximum number of dirty buffers* are set up to 1,360,000 and 1,000,000 respectively. Virtuoso maximum number of result rows is setup to 100,000, and the maximum query execution time and maximum query cost estimation are set up to 600 (seconds).

Evaluation Metrics: *i) Number of Selected Sources (NSS):* is the sum of the number of sources that have been selected per triple pattern. If the same source is selected for two triple patterns, it is counted twice by this metric. This metric measures the performance at source selection level independently of how the triple patterns are combined by the engines query decomposition. *ii) Number of Transferred Tuples (NTT):* is the sum of the number of tuples transferred from all the endpoints to the query engine during a query execution.

Additional metrics like the source selection time, execution time, answer completeness were also measured, their results are presented in Appendix B for the interested reader. Further informations (implementation, results, setups details) are available at https://sites.google.com/site/fedrasourceselection.

6. http://www.fluidops.com/fedx/, June 2015.
7. https://github.com/anapsid/anapsid, September 2014.
8. We had to implement DAW as its code is not available.
9. https://hc.apache.org/, October 2014.
10. http://www.r-project.org/
11. https://www.grid5000.fr
12. https://github.com/openlink/virtuoso-opensource/releases/tag/v7.2.1, June 2015.

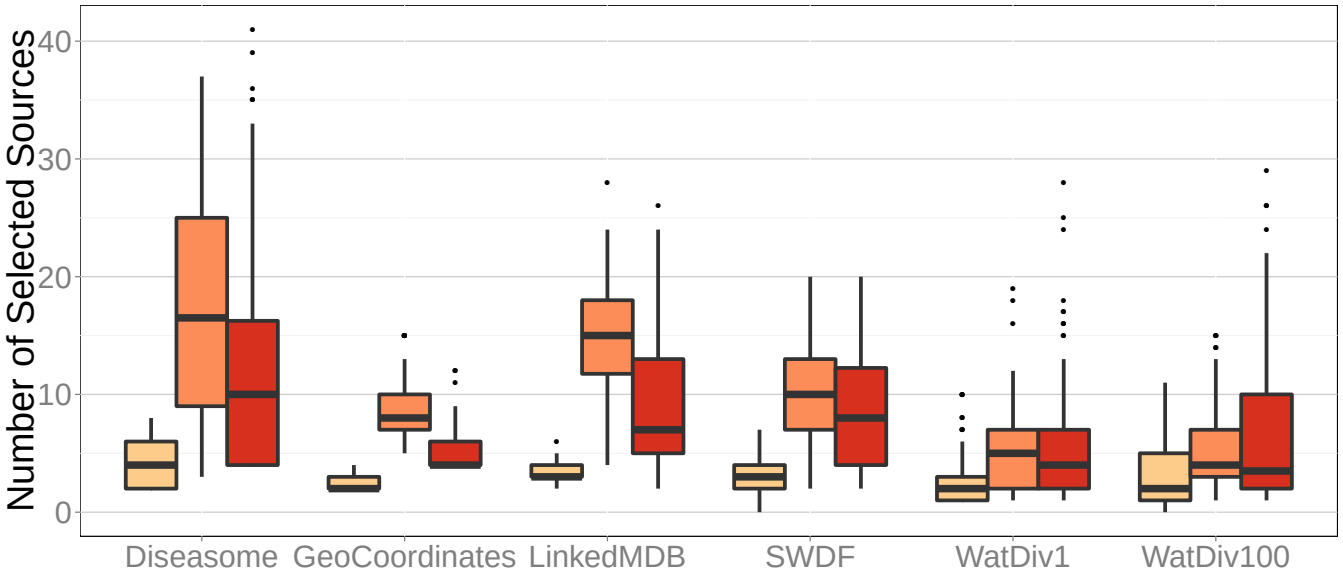


Figure 8.6 – Number of Selected Sources for execution of FEDRA+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

8.4.1 Data Redundancy Minimization

To measure the reduction of the number of selected sources, 100 queries were randomly chosen, and the source selection was performed for these queries for each federation using ANAPSID and FedX with and without FEDRA or DAW. For each query, the sum of the number of selected sources per triple pattern was computed. Boxplots are used to present the results (Figures 8.6 and 8.7). Both FEDRA and DAW significantly reduce the number of selected sources, however, the reduction achieved by FEDRA is greater than the achieved by DAW.

To confirm it, a Wilcoxon signed rank test was run with the hypotheses:

H0: FEDRA selects the same number of sources as DAW does

Ha: FEDRA selects less sources than DAW

Table 8.3 – Wilcoxon signed rank test p-values for testing if FEDRA and DAW select the same number of sources or if FEDRA selects less sources. **Bold** p-values allow to accept that FEDRA selects less sources than DAW

Federation	p-value	
	ANAPSID	FedX
Diseasome	< **2.2e-16**	**8.371e-09**
SWDF	< **2.2e-16**	**5.386e-11**
LinkedMDB	< **2.2e-16**	**5.254e-11**
Geocoordinates	< **2.2e-16**	**1.301e-05**
WatDiv1	**2.728e-13**	**1.006e-07**
WatDiv100	**4.794e-14**	**1.873e-05**

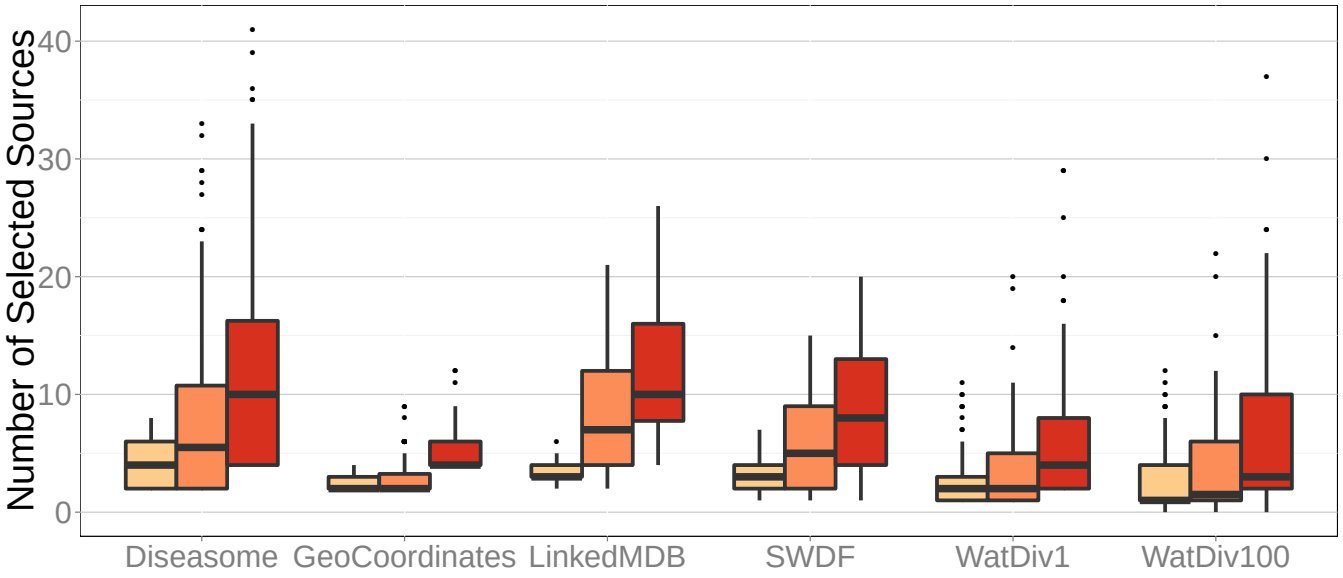


Figure 8.7 – Number of Selected Sources for execution of FEDRA+FedX ( ), DAW+FedX ( ) and FedX ( )

For all the federations and engines, p-values are inferior to 0.05. These low p-values (Table 8.3) allow for rejecting the null hypothesis that DAW and FEDRA achieved reductions are similar, and accepting the alternative hypothesis that FEDRA reduction is greater than the one achieved by DAW. FEDRA source selection strategy identifies the relevant fragments and endpoints that provide the same data. Only one of them is actually selected; in consequence, a huge reduction on the number of selected sources of up to 400% per query is achieved.

8.4.2 Data Transfer Minimization

To measure the reduction in the number of transferred tuples, queries were executed using proxies that measure the number of transmitted tuples from endpoints to the engines. Because queries that timed out have no significance on number of transferred tuples, we removed all these queries from the study. [13] Results (Figures 8.8 and 8.9) show that FEDRA source selection strategy leads to executions with considerably less transferred tuples in all the federations except in the SWDF federation and the Geocoordinates federation. In some queries of the SWDF federation, FEDRA+FedX sends exclusive groups that include BGPs with triple patterns that do not share a variable, i.e., BGPs with Cartesian products; in presence of Cartesian product, large number of transferred tuples may be generated. Queries with Cartesian products counters FEDRA positive impact over other queries.

13. Up to six queries out of 100 queries did not successfully finish in 1,800 seconds, details available at the web page.

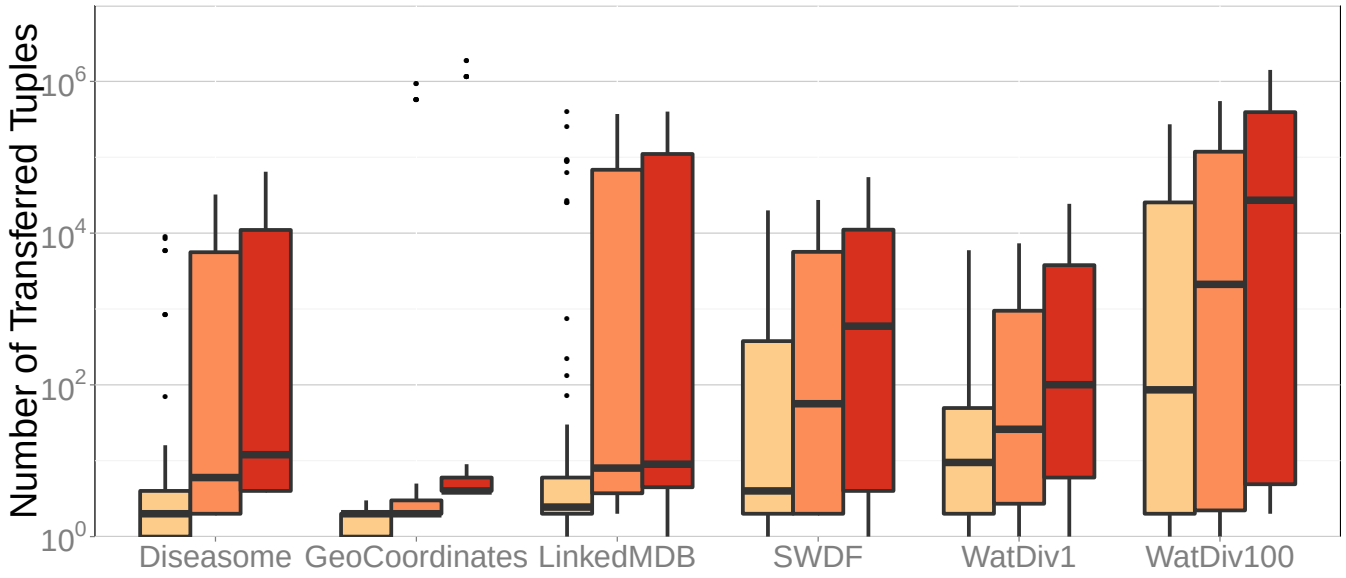


Figure 8.8 – Number of Transferred Tuples during execution with FEDRA+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

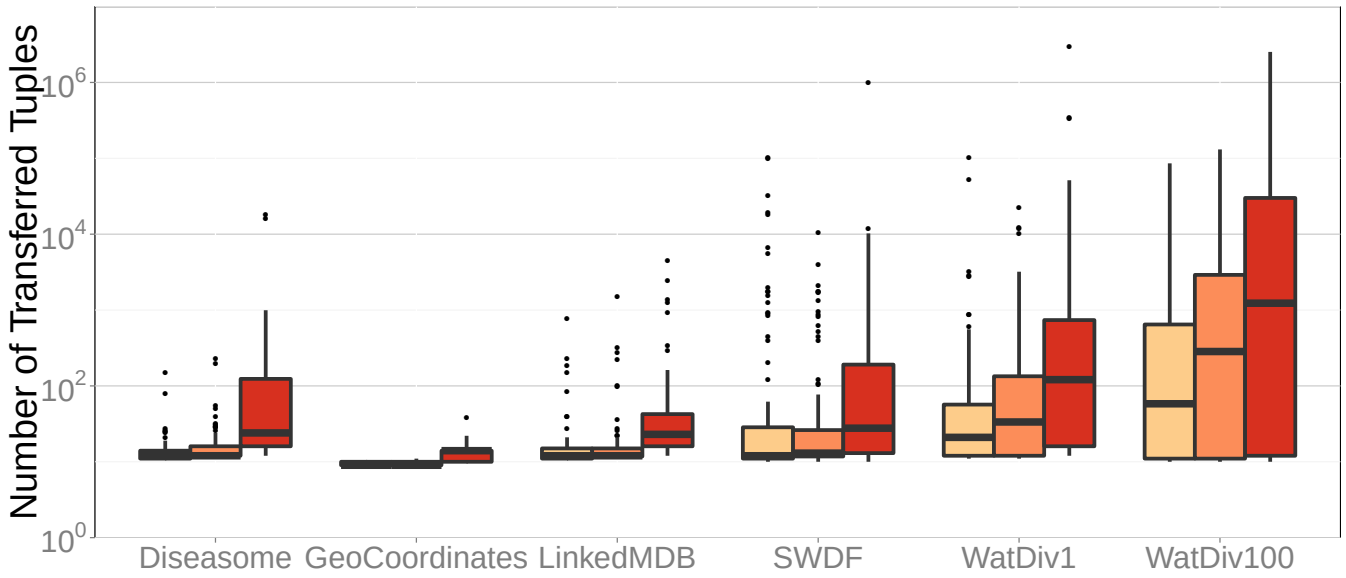


Figure 8.9 – Number of Transferred Tuples during execution with FEDRA+FedX ( ), DAW+FedX ( ) and FedX ( )

In the Geocoordinates federation the setup offers too few opportunities to execute joins in the endpoints, i.e., the different predicate combinations in the random queries considered to replicate the fragments was very narrow.

Despite that, globally FEDRA shows an effective reduction of the number of transferred tuples. To confirm it, a Wilcoxon signed rank test was run with the hypotheses:

H0: using sources selected by FEDRA leads to transfer the same number of tuples as using sources selected by DAW

Ha: using sources selected by FEDRA leads to transfer less tuples than using sources selected by DAW

Table 8.4 – Wilcoxon signed rank test p-values for testing if FEDRA and DAW transfer the same amount of data or if FEDRA transfers less data. **Bold** p-values allow to accept that FEDRA transfers less data than DAW

Federation	p-value	
	ANAPSID	FedX
Diseasome	< **2.2e-16**	**6.334e-09**
SWDF	**2.198e-11**	0.6855
LinkedMDB	**1.296e-14**	**3.427e-05**
Geocoordinates	**1.29e-12**	0.5
WatDiv1	**1.188e-07**	**1.494e-07**
WatDiv100	**2.488e-05**	**1.293e-07**

P-values (Table 8.4) are inferior to 0.05 for all federations and engines except SWDF federation + FedX engine, and Geocoordinates federation + FedX engine. In consequence, for all combinations of federation and engines except SWDF+FedX and Geocoordinates+FedX, we can reject the null hypothesis DAW and FEDRA number of transferred tuples are similar and accept the alternative hypothesis that FEDRA achieves a greater reduction of the number of transferred tuples than DAW. The reduction of the number of transferred tuples is mainly due to FEDRA source selection strategy that aims to find opportunities to execute joins in the endpoints, and mostly, it leads to a significant reduction of the number of transferred tuples of up to four orders of magnitude.

8.5 Conclusions

We illustrated how replicating fragments allow for data re-organization from different data sources to better fit query needs of data consumers. Then, we proposed a replication-aware federated query

engine by extending state-of-art federated query engine ANAPSID and FedX with Fedra, a source selection strategy that approximates SSP-FR.

Fedra exploits fragment localities to reduce the number of transferred tuples. Experimental results demonstrate that Fedra achieves significant reduction of number of transferred tuples while leading to produce complete answers.

This work opens several perspectives. First, we made the assumption that replicated fragments are perfectly synchronized and cannot be updated. We can leverage this assumption and manage the problem of federated query processing with divergence [57].

Several variants of SSP-FR can also be developed. SSP-FR does not differentiate between endpoints, and the cost of accessing endpoints is considered the same. Finally, SSP-FR and Fedra can be extended to solve the source selection problem where the number of public endpoint accesses is minimized [57].

III

Overall Conclusion and Perspectives

9

Conclusions and Perspectives.

In this thesis, we have addressed two important Semantic Web issues. First, the integration of heterogeneous sources from the Deep Web to boost the number of domains and queries that can be answered using SPARQL and Linked Data. Second, replicated data handling in the context of federated query processing against endpoint federations to improve endpoints availability.

To integrate heterogeneous sources, we have proposed SemLAV, the first scalable LAV approach that does not depend on query rewritings generation and evaluation. For querying Linked Data, query rewriting approaches are too stressed by the large number of triple patterns in SPARQL queries and the high number of sources in the Web, these characteristics prevents query rewriters to offer a practical query evaluation strategy for Linked Data. SemLAV uses query rewriters most basic information, *buckets*, to select relevant views, and ranks sources in a way that when k views have been loaded, they cover the maximal number of rewriting that can be covered with k views. In particular, our research question *in which order should the query relevant views be loaded into a graph, built during query execution, in order to use this graph to answer the query, and outperform the traditional LAV query rewriting techniques in terms of number of answers produced by time unit?* has been answered. In Chapter 5 we proposed a ranking over the views that allowed to produce answers as soon as possible, and that mostly outperforms existing query rewriting-based approaches

as shown in Section 5.3.

To handle replicated data, we have proposed FEDRA, a source selection strategy able to prune sources with replicated data in order to reduce the number of transferred tuples from the endpoints to the federated query engine. The strategy used by FEDRA gets excellent results when used in combination with ANAPSID, but results are less good when used with FedX because FedX sends Cartesian products to the endpoints. In particular, our research questions have been answered. *Can the knowledge about fragment replication be used to reduce the number of selected sources by federated query engines while producing the same answers?* In Chapter 8.2, we showed how the replicated fragments may be described, and how these descriptions can be used to find containments among the replicated fragments. These containments are used to safely prune redundant data and produce complete answers even if the number of selected sources has been highly reduced. *Does considering groups of triple patterns to be executed together, instead of individual triple patterns, produce source selections that lead to transfer less data from endpoints to the federated query engine?* In Section 8.3 we presented an algorithm that uses a set covering heuristic to evaluate as many query triple patterns as possible in the same source. This strategy allows federated query engines like ANAPSID, that reduce the number of Cartesian products sent to the endpoints, produce less transferred tuples. However, if the query engine does send Cartesian products to the endpoints, more complex strategies are needed in order to effectively reduce the number of transferred tuples.

9.1 Perspectives

The approaches presented in this thesis, and their implementations can be improved in several directions. In this section, our work perspectives are detailed.

9.1.1 Answering SPARQL queries using Linked data and Deep Web sources

The SemLAV approach implementation can be improved by loading views in parallel, and this perspective work has been partially addressed in [28], in this work parallel loading of views has been simulated loading views in blocks, and loading blocks of different views. Nevertheless, a real parallel implementation of view loading may provide even better results. Another limitation of SemLAV

approach implementation is its lack of strategies to handle memory constraints. If the memory available is not enough to store the partial RDF graph, efficient strategies should be implemented to produce a complete query using limited memory.

9.1.2 Answering SPARQL Queries against Federations with Replicated Fragments

Limitations of implementing FEDRA in FedX may be overcome with a stand-alone implementation of the FEDRA approach that transforms a plain query into a query with query decomposition and source localization represented as SERVICE clauses that avoids Cartesian products. Unfortunately, federated query engines are not yet ready to efficiently execute queries with SERVICE clauses. In this direction, we are currently working in an extended version of FEDRA that in addition to source selection also performs query decomposition, and we are also implementing new planning heuristics inside FedX that avoid Cartesian products.

Other perspective work is to use the replicated fragments hosted by various endpoints to improve federated queries performance by using them to perform parallel tasks during query execution.

Additionally, FEDRA may be extended to consider cost functions. Cost functions may be used to select the endpoints that satisfy the user criteria, e.g., in [58] public endpoint usage was reduced.

Finally, if the assumption that all the endpoints fragments are perfectly synchronized is removed, then endpoints may offer data with different values of divergence with respect the latest dataset version, in the same direction as in [57], and for instance FEDRA would have to choose the endpoints that keep the answer divergence under a certain threshold.

Results of the SemLAV experiments

A.1 Experimental study results

Table A.1 – Execution of Queries Q1, Q2, Q4-Q6, Q8-Q18 using SemLAV, MCDSAT, GQR and MiniCon, using 20GB of RAM and a timeout of 10 minutes. It is reported the number of answers obtained, wrapper time (WT), graph creation time (GCT), plan execution time (PET), total time (TT), time of first answer (TFA), number of times original query is executed (#EQ), maximal graph size (MGS) in terms of number of triples and throughput (number of answers obtained per millisecond)

Query	Approach	Answer		Time (msecs)					#EQ	MGS	Throughput
		Size	%	WT	GCT	PET	TT	TFA			(answers / msec)
Q1	SemLAV	22,660,216	33	45,434	8,322	547,310	606,697	**6,370**	15	810,638	**37.3501**
	MCDSAT	290	0	13,688	202	299,546	609,381	309,952		810,409	0.0005
	GQR	0	0	0	0	0	600,415	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,136	>600,000		0	0.0000
Q2	SemLAV	590,000	98	177,020	30,676	392,439	600,656	**260,333**	66	1,040,373	**0.9823**
	MCDSAT	0	0	15,519	105	7,058	681,246	>600,000		848,276	0.0000
	GQR	0	0	0	0	0	654,483	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,054	>600,000		0	0.0000
Q4	SemLAV	287	**100**	555,528	73,771	327	660,938	**104,501**	47	3,659,707	**0.0004**
	MCDSAT	0	0	154,451	371	181,387	601,590	>600,000		279,896	0.0000
	GQR	0	0	557,125	1,181	11,784	600,665	>600,000		84,046	0.0000
	MiniCon	0	0	413,871	650	91,136	601,750	>600,000		177,838	0.0000
Q5	SemLAV	564,220	**100**	523,084	65,333	44,102	632,809	**116,037**	28	3,396,134	**0.8916**
	MCDSAT	0	0	398,517	384	26,287	601,731	>600,000		424,431	0.0000
	GQR	0	0	0	0	0	600,481	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,132	>600,000		0	0.0000
Q6	SemLAV	118,258	59	547,763	62,896	13,291	625,173	**43,306**	24	2,931,316	**0.1892**
	MCDSAT	5,776	2	401,026	1,029	55,684	601,678	105,752		91,900	0.0096
	GQR	0	0	0	0	0	600,510	>600,000		0	0.0000
	MiniCon	3,697	1	193,817	248	51,300	637,514	418,169		2,184,680	0.0058
Q8	SemLAV	564,220	**100**	428,745	66,383	132,373	627,612	**5,393**	42	4,489,016	**0.8990**
	MCDSAT	16,595	2	403,133	576	65,935	603,297	113,211		256,382	0.0275
	GQR	1,706	0	330,065	194	31,587	607,594	272,737		1,264,385	0.0028
	MiniCon	467	0	198,384	349	271,398	616,114	166,776		1,265,295	0.0008
Q9	SemLAV	28,211	**100**	2,938	697	1,338	5,107	**1,235**	18	169,839	**5.5240**
	MCDSAT	28,211	100	5,609	445	1,643	41,505	34,392		5,417	0.6797
	GQR	28,211	100	3,310	132	1,281	5,709	1,435		5,417	4.9415
	MiniCon	28,211	100	3,086	129	1,362	5,004	862		5,417	5.6377
Q10	SemLAV	2,993,175	**100**	161,047	25,659	417,234	607,841	**9,810**	44	869,340	**4.9243**
	MCDSAT	332,488	11	19,801	67	383,421	600,000	207,191		603,769	0.5541
	GQR	0	0	0	0	0	600,639	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,138	>600,000		0	0.0000
Q11	SemLAV	2,993,175	**100**	195,950	27,442	377,255	601,042	**8,352**	43	816,308	**4.9800**
	MCDSAT	1,943,141	64	141,876	389	391,852	600,000	72,939		402,528	3.2386
	GQR	1,442,134	48	248,275	689	340,937	600,000	14,435		307,089	2.4036
	MiniCon	1,956,539	65	217,321	415	385,019	605,021	6,832		402,539	3.2338
Q12	SemLAV	598,635	**100**	258,097	41,062	303,023	609,509	**5,784**	121	1,041,369	**0.9822**
	MCDSAT	0	0	424,369	498	15,271	607,408	>600,000		509,271	0.0000
	GQR	0	0	0	0	0	600,418	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,189	>600,000		0	0.0000
Q13	SemLAV	598,635	**100**	452,288	65,043	126,345	671,893	**183,844**	124	3,509,975	**0.8910**
	MCDSAT	0	0	250,542	312	141,728	610,452	>600,000		402,531	0.0000
	GQR	0	0	36,563	344	19,757	600,376	>600,000		31,948	0.0000
	MiniCon	0	0	143,879	625	219,882	605,727	>600,000		206,689	0.0000
Q14	SemLAV	344,885	61	544,919	58,563	32,752	636,387	**29,201**	24	2,921,646	**0.5419**
	MCDSAT	10,308	1	382,674	587	63,689	614,123	133,200		1,206,075	0.0168
	GQR	0	0	0	0	0	600,714	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,319	>600,000		0	0.0000
Q15	SemLAV	282,110	**100**	471,609	63,548	109,762	645,172	**2,911**	37	3,255,223	**0.4373**
	MCDSAT	8,298	2	90,061	271	168,041	622,474	217,445		361,882	0.0133
	GQR	0	0	0	0	0	819,679	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,171	>600,000		0	0.0000
Q16	SemLAV	282,110	**100**	407,107	53,611	187,986	648,826	**2,531**	46	3,356,755	**0.4348**
	MCDSAT	8,298	2	437,590	852	32,015	601,584	103,641		74,682	0.0138
	GQR	1	0	26,460	79	94	619,761	619,702		1,136,305	0.0000
	MiniCon	252	0	110,366	181	122,022	603,821	400,416		1,151,769	0.0004
Q17	SemLAV	197,112	**100**	547,255	67,857	28,783	644,090	**1,504**	32	3,002,144	**0.3060**
	MCDSAT	156,533	79	412,525	1,727	60,858	600,067	70,476		23,192	0.2609
	GQR	45,037	22	245,953	177	350,406	600,000	27,178		1,098,117	0.0751
	MiniCon	5,779	2	262,608	361	334,810	600,001	26,952		1,099,508	0.0096
Q18	SemLAV	0	0	582,334	65,083	3,543	651,094	>600,000	12	2,806,533	0.0000
	MCDSAT	0	0	256,304	257	100,820	607,091	>600,000		411,901	0.0000
	GQR	0	0	0	0	0	600,791	>600,000		0	0.0000
	MiniCon	0	0	0	0	0	600,186	>600,000		0	0.0000

B

Fedra Experimental Study Results

B.1 Source Selection Time

To measure the source selection time, the FedX option "-planOnly" and the ANAPSID option "-p d" were used to produce only the query plan instead of executing the query. Source selection time (Figures B.1 and B.2) is the elapsed time between posing the query and obtaining the query plan, and it is measured in seconds using the system command `time`.

B.2 Execution Time

Execution time (Figures B.3 and B.4) is the elapsed time between posing the query and obtaining the query answers, it is measured in seconds using the system command `time`. A timeout of 1,800 seconds was enforced.

B.3 Answer Completeness

Answer Completeness (Figures B.5 and B.6) is the proportion of the *real* answers that are retrieved by the engine. Its value is comprised between 0 and 1. The *real* query answer is obtained executing

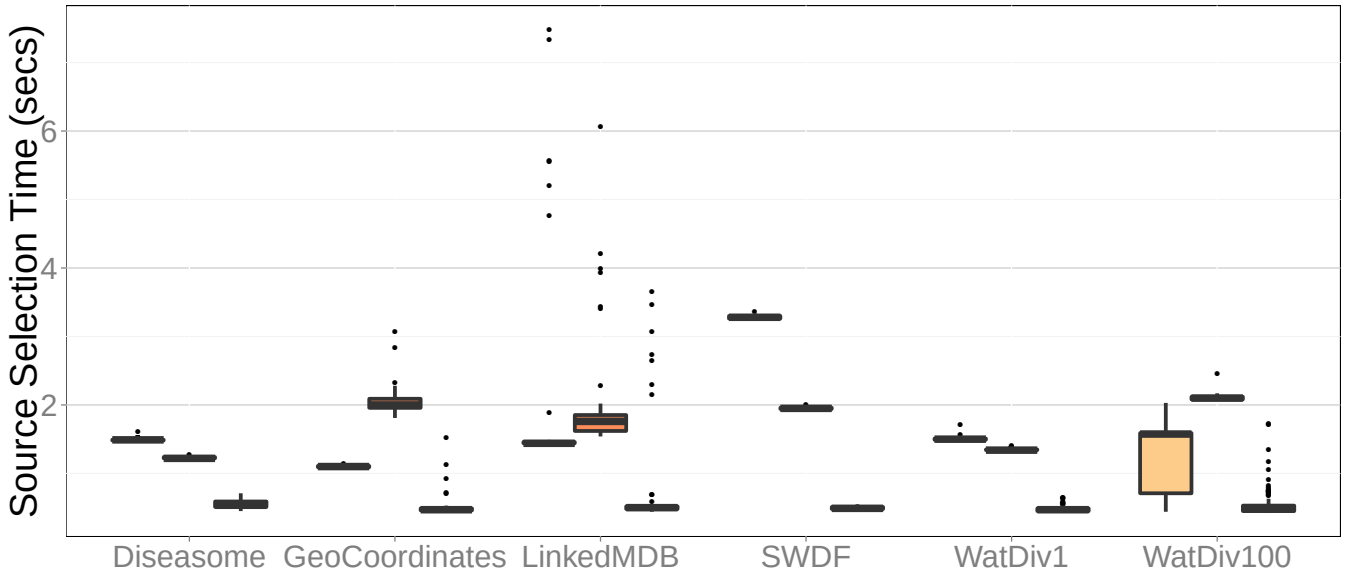


Figure B.1 – Source Selection Time (secs) for execution of FEDRA+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

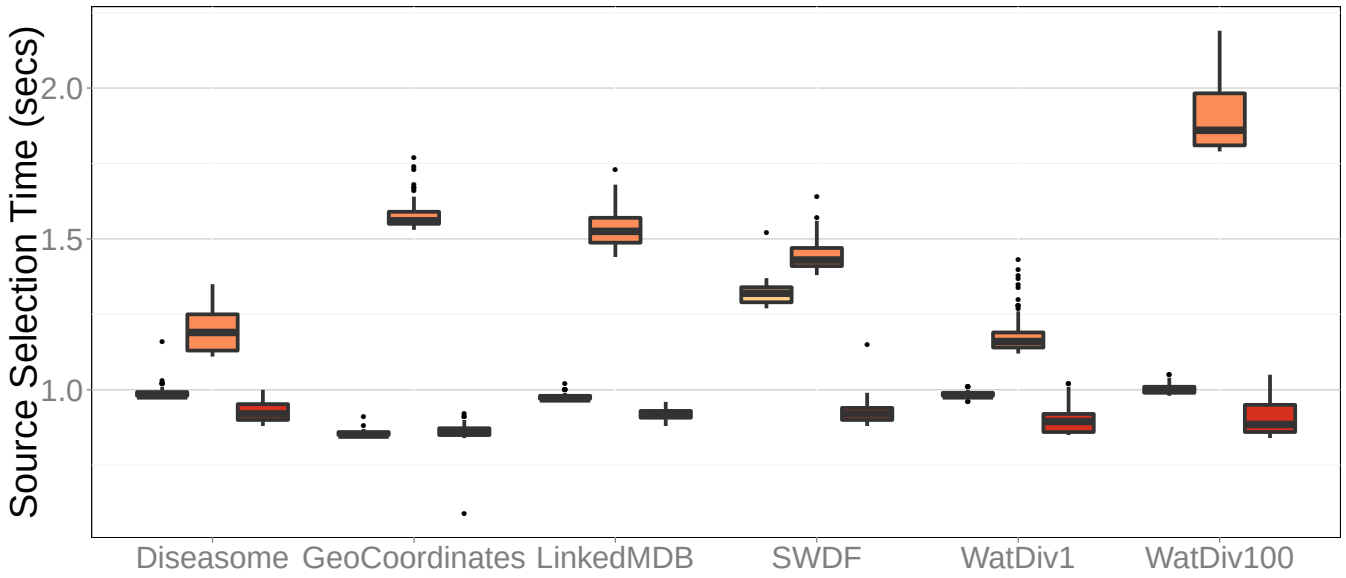


Figure B.2 – Source Selection Time (secs) for execution of FEDRA+FedX ( ), DAW+FedX ( ) and FedX ( )

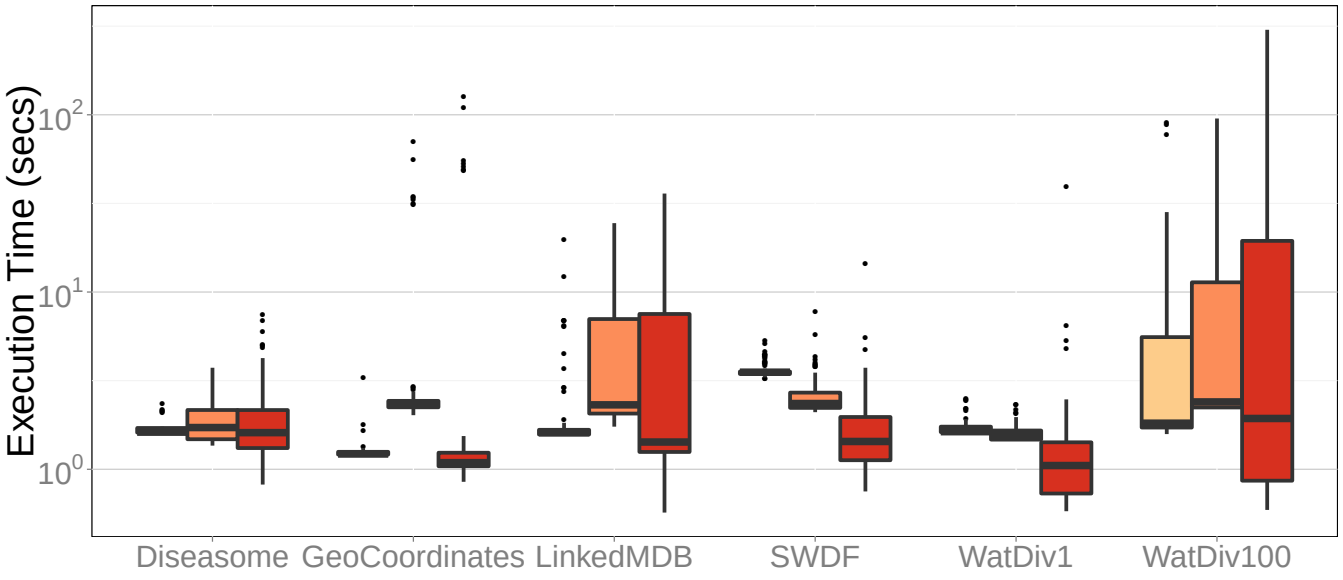


Figure B.3 – Execution Time (secs) for FEDRA+ANAPSID ( ), DAW+ANAPSID ( ) and ANAPSID ( )

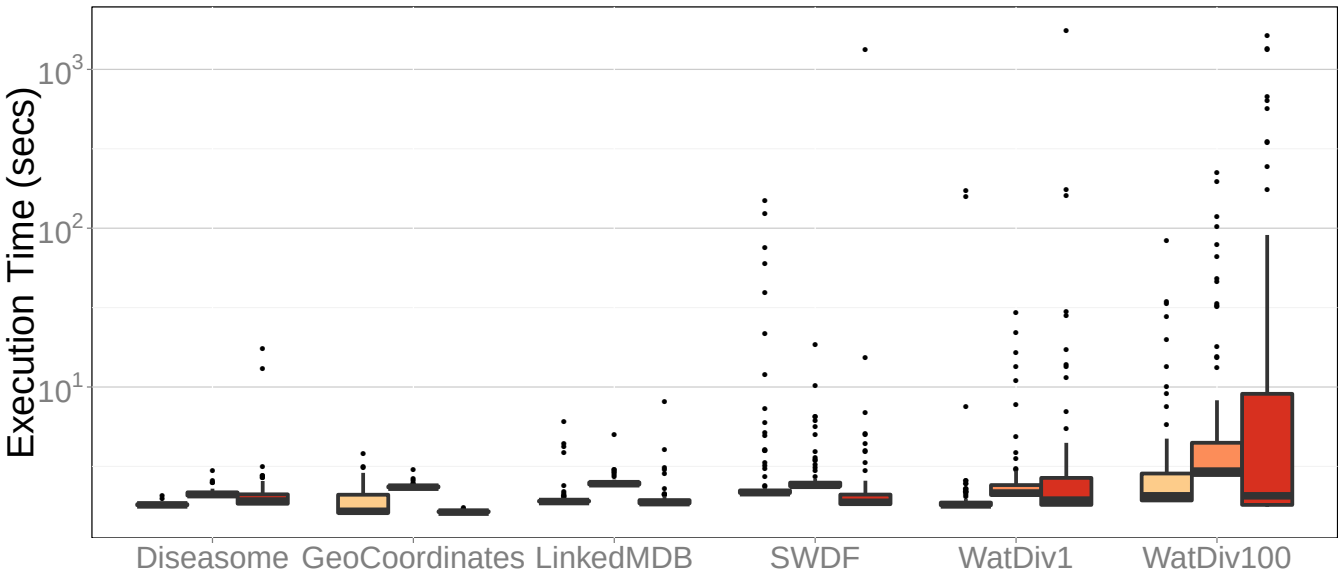


Figure B.4 – Execution Time (secs) for FEDRA+FedX ( ), DAW+FedX ( ) and FedX ( )

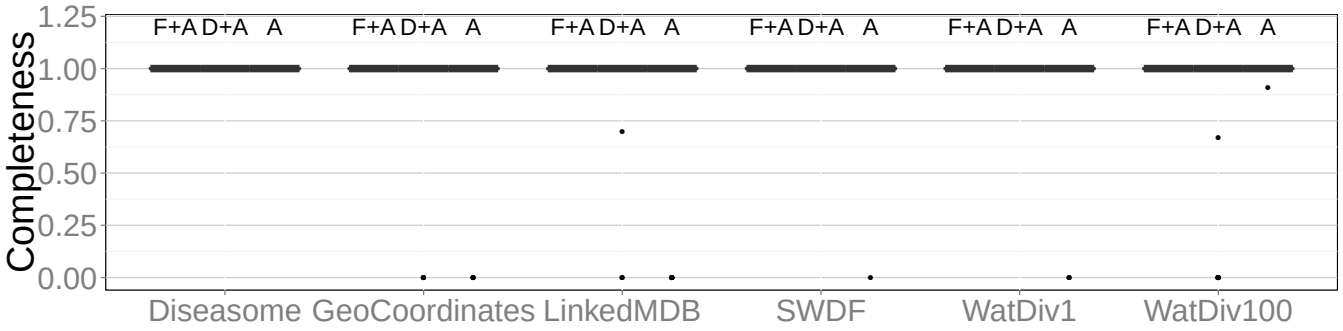


Figure B.5 – Answer Completeness for execution of FEDRA+ANAPSID (F+A,), DAW+ANAPSID (D+A,) and ANAPSID (A,) with a timeout of 1,800 secs

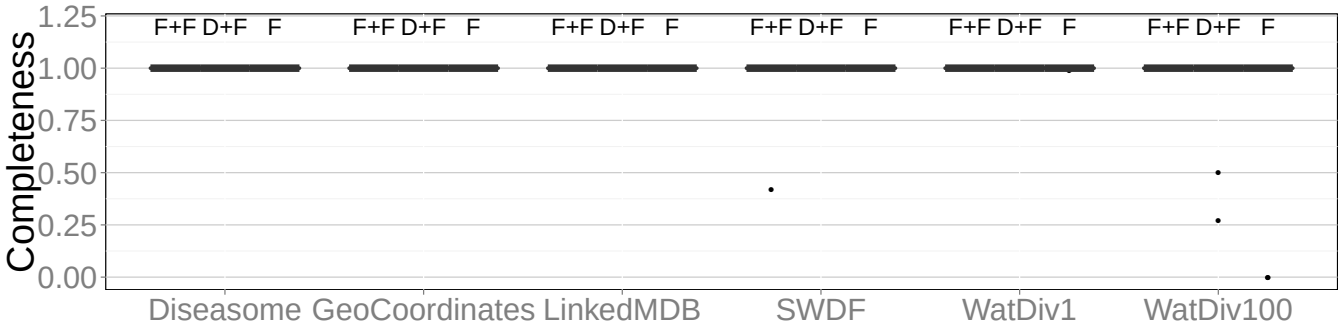


Figure B.6 – Answer Completeness for execution of FEDRA+FedX (F+F,), DAW+FedX (D+F,) and FedX (F,) with a timeout of 1,800 secs

the query against one endpoint that has all the federation data. Notice that FedX and FEDRA theoretically produce complete answers with respect to the federation data, however if there is a large number of transferred tuples during query execution, then Virtuoso endpoints may reach their maximum number of rows (100,000), and only send a partial answer to the federated query engine, and consequently the federated query engines may produce incomplete answers.

Bibliography

[1] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, and Pierre Senellart. *Web Data Management*. Cambridge University Press, New York, NY, USA, 2011. 7, 21, 39, 42

[2] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In Aroyo et al. [9], pages 18–34. 6, 8, 9, 25, 39, 63, 64, 69, 70, 72, 81, 123, 125

[3] Günes Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. Diversified Stress Testing of RDF Data Management Systems. In Mika et al. [53], pages 197–212. 91

[4] Günes Aluç, M Tamer Ozsu, Khuzaima Daudjee, and Olaf Hartig. chameleon-db: a Workload-Aware Robust RDF Data Management System. *University of Waterloo, Tech. Rep. CS-2013-10*, 2013. 91

[5] Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors. *The Semantic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29-June 2, 2011, Proceedings, Part I*, volume 6643 of *Lecture Notes in Computer Science*. Springer, 2011. 115, 116

[6] Carlos Buil Aranda, Marcelo Arenas, and Óscar Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *The Semanic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29 - June 2, 2011, Proceedings, Part II*, volume 6644 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2011. 68, 70

[7] Carlos Buil Aranda, Marcelo Arenas, Óscar Corcho, and Axel Polleres. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *J. Web Sem.*, 18(1):1–17, 2013. 68, 69

[8] Carlos Buil Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In Harith Alani et al., editors, *ISWC 2013, Part II*, volume 8219 of *LNCS*, pages 277–293. Springer, 2013. 63

[9] Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors. *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*. Springer, 2011. 111, 119

[10] Yolifé Arvelo, Blai Bonet, and Maria-Esther Vidal. Compilation of query-rewriting problems into tractable fragments of propositional logic. In *AAAI*, pages 225–230. AAAI Press, 2006. 32, 38, 45, 53, 122, 123

[11] Cosmin Basca and Abraham Bernstein. Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In Axel Polleres and Huajun Chen, editors, *ISWC Posters&Demos*, volume 658 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. 25, 39, 63

[12] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001. 5

[13] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009. 6, 21, 67, 80

[14] Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009. 23, 43, 44, 56

[15] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communication of the ACM*, 13(7):422–426, July 1970. 76

[16] Axel Börsch-Supan, Benedikt Alt, and Tabea Bucher-Koenen. 24 early retirement for the underprivileged? using the record-linked share-rv data to evaluate the most recent german pension reform. 2015. 6

[17] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-Wise Independent Permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000. 76, 77

[18] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In Ian Horrocks and James A. Hendler, editors, *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002. 70

[19] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the expressive power of data integration systems. In Stefano Spaccapietra, Salvatore T. March, and Yahiko Kambayashi, editors, *Conceptual Modeling - ER 2002, 21st International Conference on Conceptual Modeling, Tampere, Finland, October 7-11, 2002, Proceedings*, volume 2503 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 2002. 26

[20] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Query processing under glav mappings for relational and graph databases. *PVLDB*, 6(2):61–72, 2012. 22, 31

[21] Roger Castillo-Espinola. *Indexing RDF data using materialized SPARQL queries*. PhD thesis, Humboldt-Universität zu Berlin, 2012. 23, 26, 44, 56

[22] Rada Chirkova, Alon Y. Halevy, and Dan Suciu. A formal perspective on the view selection problem. *VLDB J.*, 11(3):216–237, 2002. 26

[23] Mathieu d'Aquin and Enrico Motta. Watson, more than a semantic web search engine. *Semantic Web*, 2(1):55–63, 2011. 6

[24] Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. 38

[25] Li Ding, Timothy W. Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 652–659. ACM, 2004. 6

[26] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012. 28, 30, 42, 85

[27] Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. SemLAV: Querying Deep Web and Linked Open Data with SPARQL. In *ESWC: Extended Semantic Web Conference*, volume 476 of *The Semantic Web: ESWC 2014 Satellite Events*, pages 332 – 337, Anissaras/Hersonissou, Greece, May 2014.

[28] Pauline Folz, Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Parallel data loading during querying deep web and linked open data with SPARQL. In Thorsten Liebig and Achille Fokoue, editors, *Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, PA, USA, October 11, 2015.*, volume 1457 of *CEUR Workshop Proceedings*, pages 63–74. CEUR-WS.org, 2015. 60, 102, 123

[29] Marc Friedman, Alon Y. Levy, and Todd D. Millstein. Navigational plans for data integration. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 67–73. AAAI Press / The MIT Press, 1999. 27, 31

[30] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *PVLDB*, 5(2):97–108, 2011. 26

[31] Olaf Görlitz and Steffen Staab. Federated data management and query optimization for linked open data. In Athena Vakali and Lakhmi C. Jain, editors, *New Directions in Web Data Management 1*, volume 331 of *Studies in Computational Intelligence*, pages 109–137. 2011. 67, 68, 70, 133

[32] Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In Olaf Hartig, Andreas Harth, and Juan Sequeda, editors, *COLD*, 2011. 6, 63, 69

[33] Himanshu Gupta. Selection of views to materialize in a data warehouse. In Foto N. Afrati and

Phokion G. Kolaitis, editors, *ICDT*, volume 1186 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 1997. 26

[34] Alon Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001. 30, 32, 33, 34, 35, 41, 42, 49, 85, 121, 122, 123

[35] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *WWW*, pages 411–420. ACM, 2010. 25, 39

[36] Olaf Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In Antoniou et al. [5], pages 154–169. 25, 39

[37] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the Deep Web. *Commun. ACM*, 50(5):94–101, 2007. 6

[38] Katja Hose and Ralf Schenkel. Towards benefit-based RDF source selection for SPARQL queries. In Roberto De Virgilio, Fausto Giunchiglia, and Letizia Tanca, editors, *SWIM*, page 2. ACM, 2012. 8, 64, 74, 76, 82, 87

[39] Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Olivier Corby. Col-Graph: Towards Writable and Scalable Linked Open Data. In Mika et al. [53], pages 325–340. 63, 84, 85

[40] Daniel Izquierdo, Maria-Esther Vidal, and Blai Bonet. An expressive and efficient solution to the service selection problem. In Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, editors, *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 386–401. Springer, 2010. 38, 53

[41] David S. Johnson. Approximation Algorithms for Combinatorial Problems. In Alfred V. Aho et al., editors, *ACM Symposium on Theory of Computing*, pages 38–49. ACM, 1973. 9, 90, 125

[42] Howard J. Karloff and Milena Mihail. On the complexity of the view-selection problem. In Victor Vianu and Christos H. Papadimitriou, editors, *PODS*, pages 167–173. ACM Press, 1999. 26

[43] Craig A. Knoblock, Pedro A. Szekely, José Luis Ambite, Shubham Gupta, Aman Goel, Maria Muslea, Kristina Lerman, and Parag Mallick. Interactively mapping data sources into the semantic web. In Tomi Kauppinen, Line C. Pouchard, and Carsten Keßler, editors, *LISC*, volume 783 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011. 31

[44] George Konstantinidis and José Luis Ambite. Scalable query rewriting: a graph-based approach. In Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis, editors, *SIGMOD Conference*, pages 97–108. ACM, 2011. 22, 32, 38, 45, 53, 123

[45] Donald Kossmann. The state of the art in distributed query processing. *ACM Computer Survey*, 32(4):422–469, 2000. 8, 65

[46] Günter Ladwig and Thanh Tran. Sihjoin: Querying remote and local linked data. In Antoniou et al. [5], pages 139–153. 25, 39

[47] Wangchao Le, Songyun Duan, Anastasios Kementsietsidis, Feifei Li, and Min Wang. Rewriting queries on sparql views. In *WWW*, pages 655–664, 2011. 56

[48] Maurizio Lenzerini. Data integration: A theoretical perspective. In Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis, editors, *PODS*, pages 233–246. ACM, 2002. 27, 41

[49] Alon Y. Levy. Logic-based artificial intelligence. chapter Logic-based Techniques in Data Integration, pages 575–595. Kluwer Academic Publishers, Norwell, MA, USA, 2000. 27, 29

[50] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB*, pages 251–262. Morgan Kaufmann, 1996. 32, 33, 34, 49

[51] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In Gusztáv Hencsey, Bebo White, Yih-Farn Robin Chen, László Kovács, and Steve Lawrence, editors, *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003*, pages 331–339. ACM, 2003. 6

[52] Frank Manola and Eric Miller. Resource description framework (rdf) primer. *W3C Recommendation*, 10, 2004. 5, 13

[53] Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors. *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*. Springer, 2014. 111, 115, 119

[54] Gabriela Montoya. Answering SPARQL Queries using Views. *ISWC-DC 2015 The ISWC 2015 Doctoral Consortium*, pages 33–40, 2015.

[55] Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Gun: An efficient execution strategy for querying the web of data. In Hendrik Decker, Lenka Lhotská, Sebastian Link, Josef Basl, and A Min Tjoa, editors, *DEXA (1)*, volume 8055 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2013. 39

[56] Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. SemLAV: Local-As-View Mediation for SPARQL Queries. *Transactions on Large-Scale Data- and Knowledge-Centered Systems XIII*, pages 33–58, 2014. 7, 8, 11

[57] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Fedra: Query Processing for SPARQL Federations with Divergence. Technical report, Université de Nantes, May 2014. 85, 97, 103

[58] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Efficient Query Processing for SPARQL Federations with Replicated Fragments. January 2015. 103

[59] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Federated SPARQL Queries Processing with Replicated Fragments. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference*, pages 36–51, Bethlehem, United States, October 2015. 9

[60] Gabriela Montoya, Maria-Esther Vidal, and Maribel Acosta. A heuristic-based approach for planning federated sparql queries. In *COLD*, 2012. 11, 70, 72

[61] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 3(1):37–52, 2008. 6

[62] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Springer, 2011. 63, 65, 66, 80, 133

[63] Peter F Patel-Schneider, Patrick Hayes, Ian Horrocks, et al. Owl web ontology language semantics and abstract syntax. *W3C recommendation*, 10, 2004. 5

[64] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transaction on Database Syststems*, 34(3), 2009. 17

[65] Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001. 32, 35, 36, 37, 39, 45, 53, 122, 123

[66] Eric Prud'Hommeaux, Andy Seaborne, et al. Sparql query language for rdf. *W3C recommendation*, 15, 2008. 5, 17

[67] Juha Puustjärvi and Leena Puustjärvi. The role of smart data in smart home: health monitoring case. *Procedia Computer Science*, 69:143–151, 2015. 6

[68] Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. In Sean Bechhofer et al., editors, *ESWC 2008*, volume 5021 of *LNCS*, pages 524–538. Springer, 2008. 63, 69

[69] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation. In Valentina Presutti et al., editors, *ESWC 2014*, volume 8465 of *LNCS*, pages 176–191. Springer, 2014. 74

[70] Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, Josiane Xavier Parreira, Helena F. Deus, and Manfred Hauswirth. DAW: Duplicate-AWare Federated Query Processing over the Web of Data. In Harith Alani et al., editors, *ISWC 2013, Part I*, volume 8218 of *LNCS*, pages 574–590. Springer, 2013. 8, 64, 74, 76, 78, 82, 87, 91, 92

[71] François Scharffe, Ghislain Atemezing, Raphaël Troncy, Fabien Gandon, Serena Villata, Bénédicte Bucher, Fayçal Hamdi, Laurent Bihanic, Gabriel Képéklian, Franck Cotton, Jérôme Euzenat, Zhengjie Fan, Pierre-Yves Vandenbussche, and Bernard Vatant. Enabling linked data publication with the Datalift platform. In *Proc. AAAI workshop on semantic cities*, page No pagination., Toronto, Canada, July 2012. scharffe2012a. 7

[72] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. Adoption of the linked data best practices in different topical domains. In Mika et al. [53], pages 245–260. 5

[73] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. Fedbench: A benchmark suite for federated semantic data query processing. In Aroyo et al. [9], pages 585–600. 71, 77

[74] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In Aroyo et al. [9], pages 601–616. 6, 8, 9, 22, 25, 63, 64, 69, 70, 81, 123, 125

[75] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006. 5

[76] Mohsen Taheriyan, Craig A. Knoblock, Pedro A. Szekely, and José Luis Ambite. Rapidly integrating services into the linked data cloud. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, Josiane Xavier Parreira, Jim Hendler, Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 2012. 31

[77] Dimitri Theodoratos and Timos K. Sellis. Data warehouse configuration. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB*, pages 126–135. Morgan Kaufmann, 1997. 7, 26, 39

[78] Andreas Thor, Philip Anderson, Louiqa Raschid, Saket Navlakha, Barna Saha, Samir Khuller, and Xiao-Ning Zhang. Link prediction for annotation graphs using graph summarization. In Aroyo et al. [9], pages 714–729. 6

[79] Jeffrey D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2):189–210, 2000. 21, 22, 28, 29, 121

[80] Ruben Verborgh, Olaf Hartig, Ben De Meester, Gerald Haesendonck, Laurens De Vocht, Miel Vander Sande, Richard Cyganiak, Pieter Colpaert, Erik Mannens, and Rik Van de Walle. Querying Datasets on the Web with High Availability. In Mika et al. [53], pages 180–196. 80, 84

[81] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-Scale Querying through Linked Data Fragments. In Christian Bizer et al., editors, *WWW Workshop on LDOW 2014*, volume 1184 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014. 6, 8, 80

[82] Maria-Esther Vidal, Simon Castillo, Maribel Acosta, Gabriela Montoya, and Guillermo Palma. On the Selection of SPARQL Endpoints to Efficiently Execute Federated SPARQL Queries. *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, 2015.

[83] Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadís Martínez, Javier Sierra, and Axel Polleres. Efficiently joining group patterns in sparql queries. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (1)*, volume 6088 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2010. 22

[84] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk - A link discovery framework for the web of data. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Kingsley Idehen, editors, *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009.*, volume 538 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009. 7

[85] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992. 7, 26, 39, 41, 46

[86] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics*, pages 196–202. Springer, 1992. 92

Résumé en Langue Française

Le web sémantique permet à des fournisseurs de données de mettre en ligne un nombre toujours croissant de jeux de données concernant l'ensemble de la société. Ces données peuvent être ensuite consommées en écrivant des requêtes SPARQL. Dans ce cadre, l'exécution efficace de requêtes SPARQL sur l'ensemble des données pertinentes est un enjeu crucial. Malheureusement, même si il y a un grand nombre de jeux de données dans le web sémantique, le nombre de jeux de données dans le web profond est supérieur, et SPARQL ne permet pas d'accéder aux données du web profond, restreignant considérablement les requêtes que peuvent être répondues. De plus, l'infrastructure pour exécuter les requêtes SPARQL n'assure pas une bonne disponibilité des données. Afin de traiter ces deux problèmes, nous nous sommes intéressés à l'utilisation des vues [34] dans le web sémantique afin d'optimiser l'exécution des requêtes ainsi que l'accès au web profond. Les contributions scientifiques de cette thèse sont les suivantes :

— SemLAV est un médiateur permettant d'exécuter des requêtes SPARQL sur le web profond. SemLAV s'appuie sur de vues liant les données externes au schéma global du médiateur. SemLAV évite le problème de l'explosion combinatoire de la réécriture des requêtes en calculant un ordre de matérialisation des vues concernées.

— Fedra considère une fédération de serveurs SPARQL ayant répliqués partiellement des données afin d'en améliorer la disponibilité. Cette réplication partielle peut-être considérée comme des vues. Fedra optimise l'exécution des requêtes fédérées en sélectionnant les sources de données tel que les données transférées soient minimisés.

B.4 SemLAV: requêtes SPARQL sur le web profond

La question de recherche est la suivante: *comment intégrer des sources du web des données (*linked data*) avec des sources du web profond (*deep web*) afin de répondre à des requêtes en utilisant des vues décrivant les sources comme des requêtes SPARQL conjonctives ?*

Ce problème s'inscrit pleinement dans le problème général d'intégration de données [34] avec deux approches classiques: entrepôt de données et médiateurs. Afin de ne pas bouger les données et d'avoir une plus grande fraicheur des résultats, nous privilégions l'approche basée sur les médiateurs de type *Local-As-View (LAV)* plus à même de gérer la dynamicité de nos sources [79].

Dans cette approche, les données dans les sources sont décrites en utilisant des vues du schéma global. La requête est posée en utilisant le schéma global, et elle est réécrite en utilisant les vues qui décrivent les sources. Le problème de réécriture des requêtes conjonctives est NP-Complet, et le nombre de réécritures peut-être exponentiel en fonction du nombre de sous-objectifs de la requête.

Le problème scientifique est le suivant: *Peut-on obtenir de meilleures performances en chargeant les vues dans une instance du schéma global et en exécutant la requête sur cette instance plutôt qu'en utilisant des techniques traditionnelles basées sur des réécritures de requêtes?*

SemLAV fait l'hypothèse d'absence de statistiques sur les données présentes dans les sources. Les données des sources sont décrites selon des vues basées sur requêtes SPARQL conjonctives. Le

médiateur accède aux sources du web profond par l'intermédiaire d'adaptateurs (*wrappers*). Les vues pertinentes pour une requête sont classées selon leur utilité pour la requête de telle façon que les vues les plus pertinentes sont chargées en premier. Ainsi, les chances d'obtenir des réponses très rapidement sont augmentées.

Pour illustrer l'approche SemLAV, on utilise la requête Q avec les quatre sous-objectifs présentés dans le listing 1, et un ensemble M avec cinq vues donné dans le Listing 2.

Listing 1 – Les produits, les caractéristiques et les vendeur des offres

```
SELECT * WHERE {
    ?Offer bsbm:vendor ?Vendor .
    ?Vendor rdfs:label ?Label .
    ?Offer bsbm:product ?Product .
    ?Product bsbm:productFeature ?ProductFeature .
}
```

Listing 2 – Des vues qui décrivent le contenu de cinq sources ayant des données sur des produits

```
v1(P,L,T,F):-label(P,L),type(P,T),productfeature(P,F)
v2(P,R,L,B,F):-producer(P,R),label(R,L),publisher(P,B),productfeature(P,F)
v3(P,L,O,R,V):-label(P,L),product(O,P),price(O,R),vendor(O,V)
v4(P,O,R,V,L,U,H):-product(O,P),price(O,R),vendor(O,V),label(V,L),offerwebpage(O,U),homepage(V,H)
v5(O,V,L,C):-vendor(O,V),label(V,L),country(V,C)
```

Dans l'approche traditionnelle d'une médiateur LAV, un moteur de réécriture comme Minicon [65, 34] ou MCDSAT [10] transforme Q sous la forme d'une union de 60 requêtes conjonctives définies sur la tête des vues de M. Un moteur d'exécution est ensuite chargé d'exécuter les réécritures afin d'obtenir les résultats. Dans le cas où il n'y a pas assez de ressources pour exécuter tous ces réécritures, on en exécute autant que possible avec des résultats incomplets.

SemLAV sélectionne les vues pertinentes pour Q, les classe par nombre de réécritures équivalentes puis les matérialise dans la limite des ressources disponibles. L'exécution de la requête Q sur l'instance partielle du schéma global donne des résultats équivalent à l'exécution d'un certain nombre de réécritures.

TABLE 1 – L'impact des différents ordres des vues sur le nombre de réécritures couvertes

# de vues incluses (k)	Ordre 1		Ordre 2	
	Les vues incluses (V_k)	# réécritures couvertes	Les vues incluses (V_k)	# réécritures couvertes
1	v5	0	v4	0
2	v5, v1	0	v4, v2	2
3	v5, v1, v3	6	v4, v2, v3	12
4	v5, v1, v3, v2	8	v4, v2, v3, v1	32
5	v5, v1, v3, v2, v4	60	v4, v2, v3, v1, v5	60

L'ordre dans lequel les vues sont incluses dans l'instance partielle du schéma global a un effet sur le nombre de réécritures couvertes.

Considérons deux ordres différents pour inclure les vues de l'exemple ci-dessus. *Ordre*1: v5, v1, v3, v2, v4 et v4, v2, v3, v1, v5. La TABLE 1 considère des instances partielles du schéma global faites à partir de différent nombre des vues. L'exécution de Q sur les différentes instances partielles du schéma global correspond à l'exécution d'un nombre équivalent de réécritures. Par exemple, si seulement quatre vues pourraient être incluses, *Ordre*2 correspond à l'exécution de 32 réécritures tandis que *Ordre*1 correspond à l'exécution de huit réécritures seulement. Si toutes les vues pertinentes pour la requête Q sont matérialisées alors une réponse complète sera produite. Si le nombre des vues pertinentes est grand, et que nous avons seulement des ressources pour inclure k

vues pertinentes V_k, alors nous devons inclure en premier celles qui augmentent les chances d'obtenir des réponses. En l'absence de connaissances sur la distribution de données, nous pouvons seulement supposer que chaque réécriture a les mêmes chances de produire des réponses. Ainsi, les chances d'obtenir des réponses sont proportionnelles au nombre de réécritures couvertes par l'exécution de Q sur une instance qui comprend les vues en V_k.

Nous avons comparé l'exécution des requêtes en utilisant SemLAV avec des exécutions traditionnelles utilisant les moteurs de réécriture MiniCon [65, 34], MCDSAT [10] et GQR [44]. Les résultats montrent que SemLAV améliore considérablement le temps pour obtenir les premières réponses ainsi que le débit des réponses produites par seconde (*throughput*).

En perspective, la mise en œuvre de notre approche SemLAV peut être améliorée en chargeant les vues en parallèle. Cette perspective de travail a été partiellement adressée en [28]. Dans ce travail le chargement des vues en parallèle a été simulé en chargeant les vues en blocs, et en alternant le changement de blocs de vues différentes. Néanmoins, une vrai mise en œuvre de chargement des vues en parallèle peut donner encore des meilleurs résultats. Une autre limitation de la mise en œuvre de l'approche SemLAV est la manque de bonnes stratégies pour gérer des limitations de mémoire. Si la taille de mémoire disponible n'est pas suffisante pour charger l'instance partielle du schéma global, des stratégies de gestion de mémoire efficaces peuvent être mise en place pour produire une réponse complète même avec les limitations de mémoire existantes.

B.5 Fedra: réplication des données dans le web des données liées

Le web des données liées souffre d'un problème récurrent de disponibilité des données. Les techniques de réplication de données sont traditionnellement utilisées pour pallier à ce problème. Malheureusement, nous avons observé que la réplication de données sur plusieurs serveurs détériore les performances des moteurs de requêtes fédérées. Par exemple, supposons 2 fédérations: l'une composée d'un seul serveur hébergeant les données de DBpedia [1], l'autre composée de 2 serveurs hébergeant chacun les données de DBpedia. Les temps d'exécution des moteurs de requêtes fédérées ANAPSID [2] et FedX [74] sont plus de 100 fois plus élevés pour la fédération avec deux copies par rapport à la fédération avec une seule copie. Effectivement, en l'absence de connaissances sur le schéma de réplication des données, les moteurs de requêtes fédérées n'ont d'autre choix que de contacter l'ensemble des participants pour fournir des résultats complets.

La question de recherche est la suivante: *comment exécuter des requêtes fédérées SPARQL sur des fédérations de services SPARQL avec des fragments répliqués ?*

Nous proposons de définir les fragments répliqués comme des vues enregistrées chez chacun des participants. Les moteurs de requêtes fédérés peuvent alors charger dynamiquement la définition de ces vues, reconstituer dynamiquement un schéma de réplication pour la fédération considérée et procéder à l'optimisation des requêtes fédérées dans ce cadre. La connaissance du schéma de réplication pour une fédération permet de supprimer certaines sources et de profiter éventuellement de la localité des données en fonction de requêtes.

FEDRA définit des fragments répliqués comme des vues avec un serveur d'origine. Par example, un serveur $C1$ peut répliquer un fragment de DBpedia de la manière suivante:

```
<http://dbpedia.org/sparql,
 CONSTRUCT WHERE { ?p dbprop:doctoralAdvisor ?a }>
```

Cela signifie que $C1$ a répliqué localement tous les encadrants de thèse depuis le serveur SPARQL `http://dbpedia.org/sparql`. Comme $C1$ peut également répliquer des fragments en prove-

1. http://wiki.dbpedia.org/, octobre 2015

nance d'autres serveurs, il recrée sur $C1$ une localité des données qui n'existe pas sur les serveurs d'origine.

Le problème scientifique est alors le suivant: *étant donné une requête SPARQL et un ensemble de serveurs SPARQL hébergeant des fragment répliqués, quels sont les serveurs à contacter pour sous-objectifs de la requête afin de produire une réponse complète avec un minimum de données transférées ?*

FEDRA résout ce problème en trois étapes :

1. pour chaque sous-objectif de la requête, il sélectionne les fragments pertinents et élimine les fragments redondants en utilisant les relations d'inclusion entre vues.
2. Si un sous-objectif nécessite plusieurs fragments mais qu'ils sont disponibles sur un seul serveur, alors FEDRA réduit ces fragments à un fragment virtuel.
3. Pour l'ensemble des sous-objectifs n'ayant plus qu'un seul fragment (réel ou virtuel), FEDRA sélectionne le moins de serveurs possibles en réduisant le problème en un problème de couverture par ensemble.

TABLE 2 – Les motifs de triplet de la requête $Q1$, et les fragments et les services qui sont pertinents pour répondre $Q1$

(a) Définitions des fragments pertinents pour *Q1*

F	%s% dans CONSTRUCT WHERE { %s% }
f2	?film dbo :director ?director
f3	?movie owl :sameAs ?film
f4	?movie linkedmdb :genre ?genre
f5	?movie linkedmdb :genre film_genre :14
f6	?director dbo :nationality dbr :France
f7	?director dbo :nationality dbr :United_Kingdom

(b) Localisation des fragments pertinents pour *Q1*

Les motifs de triplet		LFP	Serveurs
tp_1	?director dbo :nationality ?nat	**f6**	C1
		f7	C2
tp_2	?film dbo :director ?director	**f2**	C1,C2,C3
tp_3	?movie owl :sameAs ?film	**f3**	C2,C3
tp_4	?movie linkedmdb :genre ?genre	**f4**	C1,C3
		f5	C2

(c) Sélection des sources pertinentes par sous-objectif de la requête.

LMT	D_0(tp)	D_1(tp)	D_2(tp)
tp_1	{C1,C2}	{C1,C2}	{C1,C2}
tp_2	{C1,C2,C3}	{C1}	{C3}
tp_3	{C2,C3}	{C2}	{C3}
tp_4	{C1,C2,C3}	{C3}	{C3}
Le nombre de triplets à transférer	421.675	170.078	8.953

Considérons une requête $Q1$ composée des quatre motifs de triplet et une fédération avec trois serveurs SPARQL $C1$, $C2$ et $C3$.

La TABLE 2a montre les définitions des fragments répliqués dans la fédération, et la TABLE 2b montre pour chaque motif de triplet, les fragments répliqués pertinents ainsi que les serveurs qui les ont répliqués. Le motif de triplet $tp1$ a deux fragments pertinents : $f6$ et $f7$. Le motif de triplet $tp4$ a aussi deux fragments pertinents : $f4$ et $f5$ mais les données du fragment $f5$ sont contenus dans le fragment $f4$. Après l'étape 1 de FEDRA, seuls les fragments en gras sont candidats pour la deuxième étape.

La seconde étape de FEDRA s'intéresse à $tp1$; comme $f6$ et $f7$ sont hébergés sur 2 serveurs différents, FEDRA ne réduit pas ces fragments.

La dernière étape considère donc seulement tp_2, tp_3, tp_4. FEDRA construit un instance du problème de couverture par ensemble en essayant de couvrir l'ensemble $S = \{tp_1, tp_2, tp_3\}$ avec les sous-ensembles suivants:

1. $C_{C1} = \{tp_1, tp_4\}$,
2. $C_{C2} = \{tp_2, tp_4\}$
3. $C_{C3} = \{tp_2, tp_3, tp_4\}$

Dans ce cas, C_{c3} couvre complètement S. La sélection finale des serveurs par sous-objectif est donc la solution $D_2(tp)$ présentée dans la 2c. Cette solution offre l'opportunité au moteur de requêtes de déléguer les jointures entre tp_2, tp_3, tp_4 sur le serveur C_3, ce qui réduit considérablement le nombre de données transférées face aux autres solutions possibles $D_0(tp)$ et $D_1(tp)$.

Le premier choix, D_0, correspond à choisir tous les serveurs qui ont répliqué les fragments pertinents, ce choix produit un large nombre de données transférées. Le deuxième choix, D_1, correspond à choisir un seul serveur par fragment, ce choix produit moins de données transférées que D_0, mais ne tient pas compte de la localité des données sur $C3$. Finalement, le troisième choix, D_2, prend en compte les jointures entre les sous-objectifs de la requête, FEDRA recherche donc une sélection de serveur en tenant compte de la forme de la requête.

Nous avons implanté FEDRA, dans les moteurs de requêtes fédérées FedX [74] et ANAPSID [2], nous avons utilisé une réduction au problème de couverture par ensemble (*set covering problem*), et une heuristique existante [41] pour choisir les serveurs à sélectionner par fragment. Nous avons obtenu très bons résultats avec ANAPSID [2] et FedX [74]. Dans le cas de FedX, il existe quelques cas où les heuristiques de FedX [74] produisent des plans d'exécution avec des produits cartésiens. En conséquence, le nombre de données transférées peut être plus grand avec notre approche quand il est utilisé à l'intérieur de FedX.

Les limitations de la mise en œuvre de FEDRA à l'intérieur de FedX [74] peuvent être surmontées avec une mise en œuvre de FEDRA indépendamment du moteur de requête fédérées. Cette mise en œuvre pourrait, pour exemple, avoir comme entrée une requête simple en SPARQL 1.0 [2], et faire une transformation qui produise une requête en SPARQL 1.1 [3] avec la décomposition de la requête en sous-requêtes, et des clauses SERVICE [4] qui 'indiquent sur quel serveur chaque sous-requête doit être évaluée.

Malheureusement, les moteurs de requêtes fédérées ne sont pas encore prêts à exécuter efficacement des requêtes avec des clauses SERVICE. Dans ce sens, nous travaillons sur une extension de FEDRA qui est capable de faire la sélection de sources et la décomposition des requêtes en sous-requêtes à l'intérieur de moteurs de requêtes fédérées ANAPSID [2] et FedX [74]. Des autres perspectives de travail sont l'amélioration d'exécution des requêtes en utilisant les fragments répliqués pour réaliser des tâches en parallèle, l'utilisation des préférences de l'utilisateur comme par exemple quels sont les services qu'il préfère choisir, et incorporer des fragments qui ne sont pas parfaitement synchronises, et que pourtant ces fragments ont des divergences par rapport à la dernière version de la source.

2. http://www.w3.org/TR/rdf-sparql-query/, octobre 2015
3. http://www.w3.org/TR/sparql11-query/, octobre 2015
4. http://www.w3.org/TR/sparql11-federated-query/, octobre 2015

Contents

List of Tables

List of Figures

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Gabriela MONTOYA

Répondre aux Requêtes SPARQL grâce aux Vues

Answering SPARQL Queries using Views

Résumé

Le web sémantique permet à des fournisseurs de données de mettre en ligne un nombre toujours croissant de jeux données concernant l'ensemble de la société. Ces données peuvent être ensuite consommées en écrivant des requêtes SPARQL. Dans ce cadre, l'exécution efficace de requêtes SPARQL sur l'ensemble des données pertinentes est un enjeu crucial. Malheureusement, SPARQL ne permet pas d'accéder aux données du web profond, réduisant considérablement l'espace de recherche. De plus, l'infrastructure pour exécuter les requêtes SPARQL n'assure pas une bonne disponibilité des données. Afin de traiter ces deux problèmes, nous nous sommes intéressés à l'utilisation des vues dans le web sémantique afin d'optimiser l'exécution des requêtes ainsi que l'accès au web profond. SemLAV est un médiateur permettant d'exécuter des requêtes SPARQL sur des sources de données sur le WEB. SemLAV s'appuie sur de vues liant les données externes au schéma global du médiateur. SemLAV évite le problème de l'explosion combinatoire de la réécriture de requêtes en calculant un ordre de matérialisation des vues incriminées. FEDRA considère une fédération de serveurs SPARQL ayant répliqués partiellement des données. FEDRA optimise l'exécution de requêtes fédérées en sélectionnant les sources de données tel que les données transférées soient minimisés.

Mots clés

web sémantique, données liées, intégration de données, requêtes fédérées, sélection de sources, réplication de fragments.

Abstract

The Semantic Web allows data publishers to make available an increasing number of datasets concerning the whole society. SPARQL queries can be written to consume the datasets data. In this context, the effective execution of SPARQL queries on the relevant datasets is a critical issue. Unfortunately, SPARQL does not allow to access data from the Deep Web, and this significantly reduces the search space. In addition, the existing infrastructures to execute the SPARQL queries do not provide good data availability. To address these two problems, we have used views in the Semantic Web context to optimize the query execution and also the access to the Deep Web. SemLAV is a mediator that allows for executing SPARQL queries over data sources on the Web. SemLAV is based on views that relate external data to the mediator global schema. SemLAV avoids generating and executing an exponential number of query rewritings by computing the materialization order for the selected views. FEDRA considers a federation of SPARQL endpoints that have partially replicated datasets. FEDRA optimizes the execution of federated queries by selecting the endpoints in a way that the transferred data are minimized.

Key Words

Semantic Web, Linked Data, Data Integration, Federated Query Processing, Source Selection, Fragment Replication.