Duplicate-Aware Federated Query Processing over the Web of Data

Muhammad Saleem
Digital Enterprise Research Institute,National University of Ireland, Galway
muhammad.saleem@deri.org

Axel-Cyrille Ngonga Ngomo
Universität Leipzig, IFI/AKSW
PO 100920, D-04009 Leipzig
ngonga@informatik.uni-leipzig.de

Josiane Xavier Parreira
Digital Enterprise Research Institute,National University of Ireland, Galway
josiane.parreira@deri.org

Manfred Hauswirth
Digital Enterprise Research Institute, National University of Ireland, Galway
manfred.hauswirth@deri.org

ABSTRACT

Over the last years, the Web of Data has developed into a large compendium of interlinked data sets from multiple domains. Due to this lack of coordination, several of these data sets contain duplicate or complementary data. Several federated querying approaches have been proposed to retrieve information from these data sources. Still, only little attention has been paid to the effect of duplicated and fragmented data on the ranking of data sources during the processing of federated queries. This work presents a novel approach for federated querying based on min-wise independent permutation vectors applied to RDF predicates. Our approach begins by splitting input queries into sub-queries. Then, it processes each of the sub-queries independently. Moreover, our approach allows ranking of data sources with respect to the amount of unique relevant triples they contain while taking into account the overlap among these data sources. Thus, it can achieve sup and can therewith account for the fragmented character of the Data Web. We evaluate our approach on four datasets of different sizes by using 79 queries of various shapes. Our evaluation shows that our approach achieves significantly smaller mean squared errors that the state of the art with respect to the approximation of result set sizes as well as in the ranking of data sources. Moreover, it requires less queries to achieve the same recall as existing approaches, thus improving both execution time and network traffic.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Distributed databases*

General Terms

Algorithms, Experimentation, Theory

Keywords

Federated queries, SPARQL, Min-wise independent permutations

1. INTRODUCTION

Over the last years, the Data Web has developed into a large compendium of interlinked data sets from multiple domains [14]. One of the central principles underlying the architecture of these data sets is the reuse of URIs and vocabularies as well as the linking of knowledge bases [20]. One of the results of this architectural choice is that certain queries can only be answered by retrieving information from several knowledge bases. This type of queries, called *federated queries*, is of central importance for manifold applications such as question answering [**?**], knowledge retrieval [**?**] and data integration. Due to the reuse of URIs and the independence of the data sources, certain pieces of information (i.e., triples) can be found in several knowledge bases. For example, the name of movie directors can be found both in DBpedia and LinkedMDB. Authors of papers can be found in both the ACM and DBLP libraries. Similarly, we have noticed that the *DrugBank*[1] and *Neurocommons*[2] datasets are duplicated at *DERI health Care and Life Sciences Knowledge Base*[3]. A list of the mirrored SPARQL endpoints is given in the url[4]. An analysis of Linked Open Data Cloud shows that duplicates are most common in life sciences domain. Examples of data sets which might contain the same triples across the life science domain is shown in Figure 1. We call triples that can be found in several knowledge bases across the Web of Data *duplicates*.

[1] `http://datahub.io/dataset/fu-berlin-drugbank`
[2] `http://neurocommons.org/page/RDF_distribution`
[3] `http://hcls.deri.org:8080/openrdf-sesame/repositories/hclskb`
[4] `http://hcls.deri.org/RoadMapEvaluation/#Sparql_Endpoints`

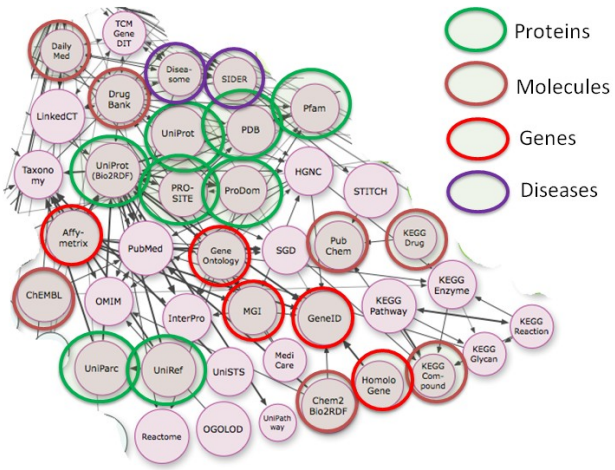


Figure 1: Possible duplicates in life sciences domain

While the importance of federated queries over the Web of Data has been stressed in previous work, the impact of duplicates has not received much attention. Yet, as we will show in the remainder of this paper, a duplicate-aware approach to query processing can lead to more time-efficient and effective algorithms for federated queries. In this paper, we address this drawback by presenting a duplicate-aware approach for query processing based on min-wise independent permutations (MIPs). By creating small summaries of the data found in the available knowledge bases, our approach is able to estimate the amount of new information contained in a knowledge base with a high accuracy, leading to a better performance with respect to source ranking, estimation of result set sizes, execution time and network traffic. An experimental evaluation on 79 queries shows that our approach outperforms the state-of-the-art by retrieving the same data with less queries. The rest of this paper is organized as follows: We begin by giving a brief overview of the state of the art in federated query processing. In addition, we argue for the use of MIPs for the approximation of result size sets. Thereafter, we present our duplicate-aware federated query processing approach. After an overview of the datasets used in this paper, we present experimental results which corroborate the superior efficiency of our approach. We conclude the paper with a discussion of our findings and an overview of future work.

2. RELATED WORK

2.1 Federated SPARQL Queries

The research work on federated query processing over the Data Web can be categorized into three main directions:

- Service description/index-assisted approaches, which make use of dataset summaries that have been collected in a pre-processing stage. These approaches may lead to a more efficient query federation. However, the index needs to be constantly updated to ensure up-to-date results. Also, the index size should not be large to ensure that it does not increase the overall query processing cost.

- Index-free approaches, in which the query federation is performed without using any stored data summaries. The data source statistics can be collected on-the-fly before the query federation. This approach promises the up-to-date records retrieval however; it may increase or decrease the query execution time, depends on the extra processing required in collecting,processing on-the-fly statistics.

- Hybrid approaches, in which some of the data source statistics are pre-stored while some are collected on-the-fly.

All of the above approaches can be further divided into two categories (a) Query federation with complete result retrieval (100% recall), and (b) Query federation with partial (sufficient) record retrieval. In our work, we focus on index-assisted query federation which can be used both for complete and partial record retrieval .

Heiner et al. [1] describe how to extend the Sesame RDF repository to support distributed SeRQL queries over multiple Sesame RDF repositories. They use a special index structure to determine the relevant sources for a query. Quilitz and Leser [2] use the theoretical knowledge of [1] to develop the first federated query engine (named DARQ) for remote RDF data sources. DARQ uses service descriptions for relevant data source selection. A service description describes the capabilities of a SPARQL endpoint. They use a query rewriting mechanism based on [8] and a cost-based optimization algorithm to reduce the query processing time with the minimum bandwidth usage. DARQ is compatible with any SPARQL endpoint that implements the SPARQL standard. Their experimental results show that the optimization algorithm can greatly reduce the query processing time.

Andreas et al. [3] propose a solution similar to DARQ using a mediator approach. The SPARQL endpoints need to register first with the mediator using HTTP POST requests with an RDF document attached. The mediator continuously monitors the SPAQL endpoints for any dataset changes and updates the service descriptions automatically. Unlike DARQ, the service descriptions remain up-to-date all time.

Olaf et al. [4] present a new approach for federated query processing over the Web of Data. Their approach discovers data that might be relevant for answering a query during the query execution itself. The discovery of relevant data is accomplished by traversing RDF links. They use an iterator-based pipeline and a URI prefetching approach for efficient query execution. Both DARQ and [3] are not able to discover relevant data sources by the query itself.

The previous techniques cannot answer all types of queries due to index or service description limitations. Umbrich et al. [5, 6], propose a Qtree-based index structure which summarizes the content of data source for query execution over the Web of Data. Their approach is able to handle more expressive queries however; because of the complete-query based source ranking, it may skip querying many sources which are capable of answering a sub-set of the query triples. Schwarte et al. [7] propose an index-free query federation for the Web of Data. Their approach gives a reasonably fast data retrieval as compared to all the previous techniques. However, since they do not keep any statistics, it works only for limited data sources.

Li and Heflin [15] build a tree structure which supports federated query processing over multiple heterogeneous sources.

They make use of an OWL reasoner to answer queries over the selected sources and their corresponding ontologies. Kaoudi et al. [16] propose a federated query technique on top of distributed hash tables (DHT). The overall goal is to minimize the query execution time and the bandwidth consumption by reducing the cardinality of intermediate results. The DHT-based optimizer make use of three greedy optimization algorithms for best plan selection.

Ludwig and Tran [18] propose a mixed query engine that assumes some incomplete knowledge about the sources to select and discover new sources at run time. A Symmetric Hash Join is implemented to incrementally produce answers. This approach is extended to query both remote and local linked data [19].

Maribel et al. [21] present ANAPSID, an adaptive query engine that adapts query execution schedulers to SPARQL endpoints data availability and run-time conditions. ANAPSID provides physical SPARQL operators that detect when a source becomes blocked or data traffic is bursty. The operators produce results as quickly as data arrives from the sources. Amol et al. [17] present a survey of the adaptive query processing techniques, common issues, and the settings in which each piece of work is most appropriate. Avalanche [24] gathers endpoints dataset statistics and bandwidth availability on-the-fly before the query federation. The advantage of this technique is the live query execution over the Web of Data. However, on-the-fly statistics calculation can increase the overall query execution time. Olaf [25] propose a heuristics-based technique to minimize query intermediate results.

Other noteable contributions such as LIMES [26], SILK [27], EAGLE [28] provides time-efficient approaches for link discovery using various distance measure techniques. The LIMES framework utilize the mathematical characteristics of metric spaces to compute similarity estimates between instances. It makes use of the triangle inequality to partition the metric space into different portions. Each of these portions of the space is then represented by an exemplar [33] that allows to compute an accurate distance approximation between different instances. SILK implements a multi-dimensional blocking approach for link discovery. The Levenshtein distance [34] is used for approximate for similarity estimation between different instances. EAGLE presents a novel active learning approach to learning link specifications based on genetic programming. EAGLE generates highly accurate link specifications while reducing the annotation burden for the user.

Few other contributions address the problem of SPARQL query federation [22], [23] but none, to the best of our knowledge has taken into account the possibility of mutual overlap between different RDF datasets before query execution. Similar to DARQ engine, we use service descriptions for query federation. In addition, we store MIPs vector statistics for each of the indexed data source. Our query federator makes use of this special statistic information to estimate the possible overlap between the results of different sub-queries in order to avoid the retrieval of redundant information.

2.2 Dataset Overlap Estimation

Avoiding the retrieval of duplicate records is a crucial issue in large scale distributed datasets. As the SPARQL endpoints are autonomous, there is always the possibility of high mutual overlap between different datasets residing in different endpoints. It is thus likely that an optimized query execution plan contains a sub-query that retrieves records which are already retrieved by another sub-query from a different SPARQL endpoint. In this case it is not necessary to execute the later sub-query, as it would not only result in duplicate records being retrieved but also increase the total processing time.

Statistical synopsis like min-wise independent permutations [9], bloom filters [10], hash sketches [36], XSKETCH [37], fractional XSKETCH [38], and compressed bloom filters [11] have been extensively used in the literature to provide a compacted representation of data sets. Statistical synopsis allow to estimate a number os set operations such as overlap and union without having to compare the original sets directly. For example, Bender et al. [12] propose a technique for predicting the query-specific mutual overlap datasets in a peer to peer network. MIPs have proved to be the more accurate overlap estimation with small bandwidth consumption[9] therefore we have chosen them in our approach.

3. NOTATION

In this section, we present the core of the notation that will be used throughout this paper. We denote data sources with S and the total number of data sources with n. The set of all data sets is dubbed $\mathfrak{S}$. The set of all possible result sets is denoted R while the set of all possible SPARQL queries is labeled with Q. A data source ranking function $rank : S \times Q \rightarrow \{1 \dots n\}$ is a function that assigns a ranking to each data source given a particular query $q \in Q$. Note that for any source ranking function $rank$, we assume that $\forall S, S' \forall q \in Q : S \neq S' \iff rank(S,q) \neq rank(S',q)$. A result set estimation function $est : S \times Q \rightarrow \mathbb{N}$ aims at approximating the size of the result set that will be returned by a given query. Note that this function plays a crucial role in the processing of federated queries as it is most commonly used to decide upon the ranking of data sources for a given query. The aim of a federated query system such as the one described in this work is thus to optimize its estimation function est so as to ensure a ranking of the source close to the optimal ranking.

4. DUPLICATE-AWARE FEDERATED QUERY PROCESSING

4.1 Overview

Given a query, it is first parsed, then sent to federator which makes use of the services descriptions directory/index to decompose the query into duplicate free multiple sub-queries. Each sub-query is optimized to generate an optimized execution plan. The optimized sub-queries are forwarded to the relevant SPARQL endpoints. The results of each sub-query execution are integrated and the final query result set is generated. This architecture for duplicate-aware federated query processing is shown in Figure 2. While this architecture is similar to other federated query systems, the federator in our system differ significantly from those of other approaches as it makes use of MIPs statistical information to generate duplicate-free query execution plans. The generation of MIPs statistical information is explained in the next section.

4.2 Service Description Directory

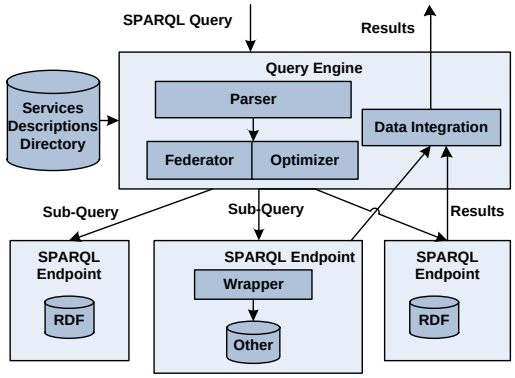


Figure 2: Duplicate-aware federated query processing architecture

To find the set of relevant servers and generate the appropriate sub-queries, the federator needs information about the different data sets available at the different servers. We call the catalog containing such information the *service description directory*. Each service description provides a declarative description of the data available from a server along with statistical information.

For each server our *service description directory* stores *server url, graph name*, and for each distinct predicate p we record *predicate name, MIPs vector, subject selectivity, object selectivity*, and *total number of triples* having predicate p. The MIPs vectors are used to estimate the dataset overlap among the capable datasets before the sub-query distribution and it will be explained in detail in section 4.3. The *subject* and *object selectivity* are used to estimate the sub-query result set when *subject, object* are bound, respectively. We call the set $C_i = \{(p_i, mipsv_i, sbjsel_i, objsel_i, totTrpl_i)\}$ as a *capability* of the data source. The total number of capabilities of a data source is equal to the number of distinct predicate in that data source. A sample service description is given in the Listing 1.

It is very crucial to keep the directory size small to minimize the pre-query federation time. However,this directory must contain sufficient information to enable the generation of sub-query, the optimization, and the pre-processing dataset overlap estimation. Our index size is mostly depended upon the size of the MIPs vectors which can be adjusted to any length. In general MIPs can provide a good estimation of the overlap between sets with a few integer in length. Min-Wise Independent Permutations have been in the peer to peer networks literature [13, 39, 40].

4.3 Min-Wise Independent Permutations

MIPs assumes that the set elements can be ordered and computes N random permutations of the elements. Each permutation uses a linear hash function of the form $h_i(x) := a_i * x + b_i \ mod \ U$ where U is a big prime number and a_i, b_i are fixed random numbers [13]. By ordering the resulting hash values, we obtain a random permutation. For each of the N permutations, the MIPs technique determines the minimum hash value, and stores it in an N-dimensional vector, thus capturing the minimum set element under each of these random permutations. The technique is illustrated with an example in Figure 3. Its fundamental rationale is that each element has the same probability of becoming the

Listing 1: A Service Description Example

```
[] a sd:Service ;
     sd:url    <http://localhost:8890/sparql> ;
     sd:graph ''diseasome/DS1'';
     sd:capability
     [
      sd:predicate  diseasome:name ;
      sd:MIPv  ''-6908232 -7090543 -6892373 -7064247 ...'';
      sd:subjectSelectivity  ''0.0068'' ;
      sd:objectSelectivity  ''0.0069'' ;
      sd:triples    147  ;
      ] ;
     sd:capability
     [
      sd:predicate  diseasome:chromosomalLocation ;
      sd:MIPv  ''-7056448 -7056410 -6845713 -6966021 ...'';
      sd:subjectSelectivity  ''0.0062'' ;
      sd:objectSelectivity  ''0.0072'' ;
      sd:triples    160  ;
      ] ;
```

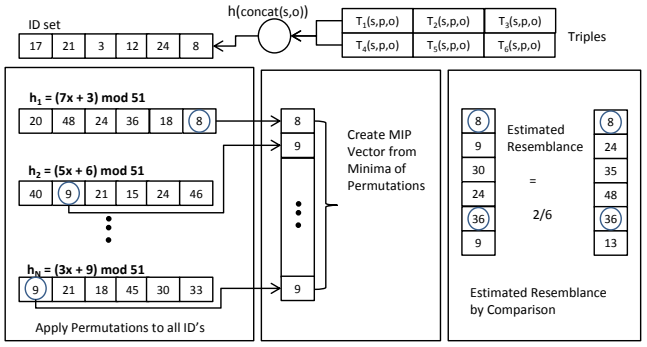


Figure 3: Min-Wise Independent Permutations

minimum element under a random permutation. By using sufficiently many different permutations, we can approximate the set cardinality.

Let S_A and S_B be the two MIPs set, the cardinality of the intersection is defined as the overlap between these two sets. The overlap and union of the two sets is given in Equation 1 and Equation 2, respectively.

$$Overlap(S_A, S_B) = |S_A \cap S_B| \tag{1}$$

$$Union(S_A, S_B) = |S_A \cup S_B| \tag{2}$$

An unbiased estimate of the pair-wise resemblance of the sets can be obtained by counting the number of positions in which the two MIPs vectors have the same number and dividing this by the number of permutations N. Essentially, this holds as the matched numbers are guaranteed to belong to the intersection of the sets. The resemblance formula is given in Equation 3.

$$Resemblance(S_A, S_B) = \frac{Overlap(S_A, S_B)}{Union(S_A, S_B)} \tag{3}$$

In our service descriptions the MIPs are used as follow: for a predicate $p \in S$, we create a MIPs vector representing all triples $T \in S$ having predicate p. We concatenate the *subject* and *object* of each triple in T and apply a hash function to get the *ID set*. Finally, we computes N random permutations of the *ID set*. The resulting MIPs vector is stored against each capability of the service description. Note that we are not applying hash function to the complete triple because the predicate is already stored with each service capability and source selection is bound by query predicate.

4.4 Source Selection

A SPARQL query contains one or more basic graph patterns. Similar to DARQ, our query planning step is performed separately for each basic graph pattern. The source selection algorithm used in DARQ for a query simply matches all triple patterns against the capabilities of the data sources. The matching compares the predicate in a triple pattern with the predicate defined for a capability. Let $S = \{(s_1, C_1), ..., (s_n, C_n)\}$ be a set of all indexed data sources $s_1..s_n$ and their capabilities $C_1..C_n$, where $C_i = \{(p_i, mipsv_i, sbjsel_i, objsel_i, totTrpl_i)\}$.

Let BGP be a set of triple patterns in a filtered basic graph pattern. The result of the source selection is a set of data sources D_j for each triple pattern $t_j = (s_j, p_j, o_j) \in BGP$ such that

$$D_j = \{\text{s}|(\text{s,C}) \in \text{ S} \wedge \exists \ p_j \in \text{ C }\}$$

We extend the DARQ source selection algorithm by comparing MIPs vectors for duplicates. For a triple pattern, we compare the MIPs vectors of all the relevant data sources for possible duplicates. A specific source s is only added to the set of capable sources D_j, if it is able to produce a number of new results above a specific threshold value predefined for final source selection. For a triple $t \in BGP$ and a dataset $s \in S$, if the value of new results generated is less than the threshold, s is ignored from set D_j. For 100% recall the threshold value should be 1. Which means if a source is able to produce a single triple, it will be added to the set of capable sources. For sufficient (not complete) record retrieval the threshold can be fixed to any number greater than 1. For all of the 79 queries in this work, our threshold value is 1.

4.5 Source Ranking

4.5.1 Triple Pattern Source Ranking

Algorithm 1 shows the single triple pattern source ranking. For a triple *t(s,p,o)* $\in T$ the set of capable data sources D is obtained using the source selection algorithm explained in previous section. The source $d \in D$ which has the highest *totTrpl* value in the matched capability *C(p,mipsv,sbjsel,objsel,totTrpl)* is selected as first in the rank. The union and intersection of *MIPs Vectors* is explained in section 4.3. In order to get the accurate result set estimate, we multiply *sbjsel, objsel* $\in C$ to the *orignal size* of the *MIPs Vectors* if *subject* or *object* of the triple pattern is bound respectively. The same is true for rank 1 dataset selection. However, the multiplication is performed with *totTrpl* of mached capability instead of *orignal size*.

It is important to note that the number of elements in the MIPs vector may be different from its *orignal size*. The *orignal size* is the number of elements in the *ID set* over which we applied N random permutations to get the MIPs Vector as explained in MIPs section. In the line 24 we set the threshold value to 1 in order to get 100% recall. For each triple pattern $t_j = (s_j, p_j, o_j)$ in the query $q \in Q$, we apply the Algorithm 1 to get a set of selected capable data sources D_{s_j} which is used in sub-query generating algorithm explained in Section 4.6.

4.5.2 Join Source Ranking (this section is not well written)

Similar to DARQ, we perform individual triple based sub-query federation. Consequently, triple pattern source ranking is most important for sub-query escaping. The advantage of this approach is that if a data source is not capable of answering the complete query, it may produce enough results for a specific sub-set of query triples. For the comparison purpose we propose a complete query source ranking which works as follow: For a star shaped queries, the overall result set is bound by the triple with lowest number of estimated records. Therefore, we perform the source ranking for that triple (using Algorithm 1) and use this as complete query ranking. The lowest records query triple can be estimated using the rank 1 source selection explained in previous section. However, this time we are looking for a source which has the lowest estimated value of all. For path shaped queries the total records are bound by the triple which has *subject* variable equal to the *object* of the previous query triple. For example the result set in the query given in Listing 2, is bound by the second triple in which the *object* variable in the first triple is used as *subject* in the second triple. Therefore, we perform the triple pattern ranking for second triple and use this as ranking for complete query.

Algorithm 1 triple pattern source ranking

Require: t(s,p,o) ∈ T //triple pattern
1: D = getCapableSources(t) //all sources capable of answering triple t
2: $rank_1Source$ = getMaxSizeSource(D,t)
3: rnkNo = 1
4: unionMipv = $rank_1Source$.getMIPv(t)
5: D_s[rnkNo] = selectedSource
6: D = D - {selectedSource}
7: rnkNo = rnkNo+1
8: **while** $D \neq \emptyset$ **do**
9: selectedSource = null
10: maxDistinctRecords =0
11: **for** each $d_i \in$ D **do**
12: mipv = d_i.getMipv(t)
13: **if** s is bound in t **then**
14: mipv.orignalSize =mipv.orignalSize*d_i.getSbjSel(t)
15: **else if** o is bound in t **then**
16: mipv.orignalSize =mipv.orignalSize*d_i.getObjSel(t)
17: **end if**
18: overlap = unionMipv.intersectionSize(mipv)
19: distinctRecords = mipv.getOrignalSize()-overlap
20: **if** distinctRecords > maxDistinctRecords **then**
21: selectedSource = d_i
22: maxDistinctRecords = distinctRecords
23: **end if**
24: **end for**
25: **if** maxDistinctRecords $\geqslant$ 1 **then**
26: D_s[rnkNo] = selectedSource
27: selectedMipv = selectedSource.getMipv(t)
28: unionMipv = unionMipv.union(selectedMipv)
29: rnkNo = rnkNo+1
30: **end if**
31: D = D - {selectedSource}
32: **end while**
33: **return** D_s //rank wise selected sources

Listing 2: A P-1 query example

```
SELECT ?s
  WHERE
```

Algorithm 2 sub-query generation

Require: T = { $t_1, .., t_n$}
S = {D_{s_1},..,D_{s_n}} // sets of data sources obtained from algorithm 1

```
1: sub-queries = ∅, seperateSub-queries = ∅
2: for each t_i ∈ T do
3:   if D_{s_j} = {s} then
4:     q = queries.getQuery(s)
5:     if q not null then
6:       q.T = q.T + t_i
7:     else
8:       sub-queries = sub-queries + ({t_i},s)
9:     end if
10:  else
11:    for each s_j ∈ D_{s_j} do
12:      seperateSub-queries = seperateSub-queries + ({t_i},s)
13:    end for
14:  end if
15: end for
16: return sub-queries ∪ seperateSub-queries //Return all sub-queries
```

```
{
    ?s  <p1> ?o .
    ?o  <p2> ?o1
}
```

4.6 Sub-query generation

The sub-query generation algorithm is similar to the one presented in DARQ. However, instead of using all capable sources, we use selected data sources (obtained from Algorithm 1) for the sub-query generation. Let *(T,s)* be a sub-query, where T is a set of triple patterns, and s is the data source that can answer this sub-query. The sub-query generation is shown in Algorithm 2. If a triple pattern t_j matches exactly one data source *($D_{s_j} = \{s\}$)* then the triple will be added to the set of sub-queries for this data source. All triples in this set will be sent to the data source in one sub-query. If a triple matches more than one data sources then the triple will be sent individually to all matching data sources in separate sub-queries. After the sub-queries are generated, the federator sends all of them to the optimizer for further optimization, before the federation. Our optimizer and data integrator module is similar to DARQ.

5. EXPERIMENTS AND RESULTS

The goal of our experiments was to quantify how well our duplicate-aware solution approximates the result sets from each of the data source from which it can choose and how this approximation affects the way the data sources are ranked. For this purpose, we compared our approach with two other approaches: DARQ and the optimal duplicate-unaware solution to result set estimation and query ranking. In the following, we describe our experimental setup in detail and contrast the results achieved by each of the approaches. All data used is either publicly available or can be found at the project webpage.[5]

[5] `http://???`

Dataset	BGP	S-1	S-2	P-1	P-2	P-3	Total
Diseasome	5	5	5	4	5	2	26
Geo	5	5	5	0	0	0	15
LinkedMDB	5	0	0	0	0	0	5
Publication	5	5	5	7	7	4	33
Total	20	15	15	11	12	6	79

Table 2: Distribution of query types across datasets

5.1 Experimental Setup

5.1.1 Datasets

Within our experiments, we used the four datasets described in Table 1. We chose data sets of different data sizes to gain some insights on the scalability of our approach. The size of the MIPS vectors was set to achieve a compression ratio of approx. 98% greater than the previous work [5]. In addition, we emulated the difference between the number of relevant triples across datasets by introducing the discrepancy constant d. Given n data sources of sizes $|S_i|$

$$d = \max_{1 \leq i \leq n} |S_i| - \min_{1 \leq j \leq n} |S_j|. \quad (4)$$

Note that the discrepancy accounted for between ca, 1.6% and 2.8% of the triples and was to increasingly higher values for increasingly larger data sets. The number of slices was set to 10 in all experiments.

5.1.2 Queries

We used three main types of queries. Simple basic graph pattern (BGP) queries consisted of exactly one triple pattern in the query. Star-shaped and path-shaped queries were defined as in [5]: star-shaped queries (denoted S-k) have one variable as subject and contained k triple patterns. Path-shaped queries, denoted P-k on the other hand were generated by using a random walk approach to generate a path of length k. Note that previous work has shown that these query shapes are the most common shapes found in real-world RDF queries [35]. Overall, our benchmark data consisted of 79 queries as shown in Table 2. Note that some query shapes could not be used on certain datasets due to the topology of the ontology underlying the datasets not being permissive for such queries. For example, S-2 queries could not be sent to LinkedMDB because

5.1.3 Metrics

We used two measures to compare our approach with the state of the art. First, we measured the mean squared error of each approach with respect to their result set evaluation as follows: For each query q_i, each source S_i contained a number $R_i(q_i)$ of non-duplicate results. Each of the approaches generated an approach $\tilde{R}_i(q_i)$ of $R_i(q_i)$. The mean squared error with respect to the result set size (short $RSMSE$) was then computed as follows:

$$RSMSE = \frac{1}{|Q|} \sum_{q_i \in Q} |R_i(q_i) - \tilde{R}_i(q_i)|^2. \quad (5)$$

In addition to the MSERS, we also computed how well the approaches were able to rank the data sources with respect to the unseen triples they would return for a given query. Given the ideal ranking $rank(S_i, q_j)$ of each data source S_i for each query q_j, we computed the mean squared ranking error (short $RankMSE$) of an approach characterized by the

Dataset	Size (MB)	Summary Size (MB)	CR (%)	Slices	Discrepancy	Duplicate Slices	Duplicate Datasets	Total Triples	Sum. Gen. Time (sec)
Diseasome	18.6	0.64	96	10	1,500	1	10	91,122	9
Publication	39.0	1.15	97	10	2,500	1	10	234,405	16
Geo Coordinates	274.1	4.66	98	10	50,000	2	5,8	1,900,006	1302
LinkedMDB	448.9	7.9	97	10	100,000	1	2	3,579,616	1837

Table 1: Overview of datasets. CR stands for Compression Ratio.

the ranking function $\tilde{r}(S_i, q_j)$ by using the following formula:

$$RankMSE(q_j) = \frac{1}{|\mathfrak{S}|} \sum_{S_i} |rank(S_i, q_j) - \tilde{r}(S_i, q_j)|^2. \quad (6)$$

$$RankMSE = \frac{1}{|Q|} \sum_{q_j \in Q} RankMSE(q_j). \quad (7)$$

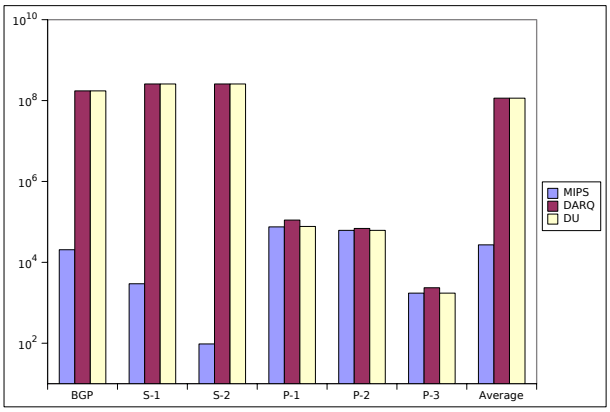
For both measures, higher values are obviously worse.

5.2 Results

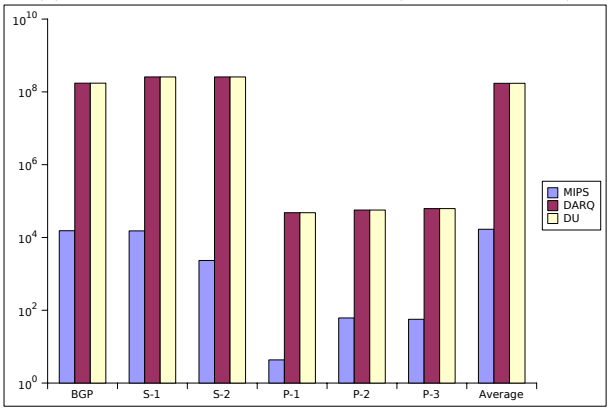
5.2.1 Result set estimation and ranking

The RSMSE results are shown in Figures 4 while the ranking results are shown in Figure 5. In each figure, we compared the results of the approaches w.r.t. both the whole input query and to each of the triples patterns contained in the input queries. Our results clearly show that we outperform the state of the art w.r.t to both result set estimation and ranking. Especially, our estimation of the result set sizes is approximately 4 orders of magnitude better than DARQ's or even DU's. This is due to our approach being able to capture the existence of duplicates in the data sets and thus to better approximate the number of new results that can be expected from an endpoint. As expected, the ranking error for datasets decreases when carrying the ranking of data sources at triple pattern level instead of query level (see Figures 4b and 4a). This is simply due to the overall error increasing when combining the several triple patterns to a query. The same overall behavior is also displayed when considering the result set estimation error (see Figures 5b and 5a). Interestingly, the approximation of our approach is slightly poorer on star-shaped queries (approx. 0.2 on S-2 queries). This is a result of our approach to generating the estimations for whole query, which is carried out by using the lower bound of the estimation of single query patterns in the query. Small errors in the estimation of the size of the result size can thus lead to minor absolute errors.

As stated before, we used a compression ratio of approx. 98% across all experiments. The main advantage of such a high compression ratio is the small size of the index. Yet, the higher the compression ratio, the larger the error margin of the ranking function. When given a query whose sub-queries lead to very similar result set estimations, this larger error margin can lead to substantial $RankMSE$ and $RSMSE$ rates. We thus studied the effect of decreasing the compression ratio on the $RSMSE$ and $RankMSE$ for such a query. Our results in Table 3 suggest that MIPS vectors can successfully detect even small differences in the sizes of the result sets when given a sufficiently small compression ratio.



(a) RSMSE for query-wise selection (vertical log scale)



(b) RSMSE for subquery-wise selection (vertical log scale)

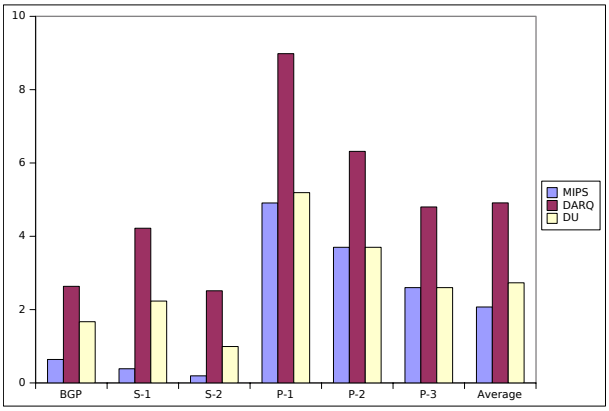
Figure 4: Mean squared errors with respect to result set evaluation

Compression Ratio	$RSMSE$	$RankMSE$
97%	253.9×10^6	3.4
94%	33.3×10^6	0.6
91%	0	0

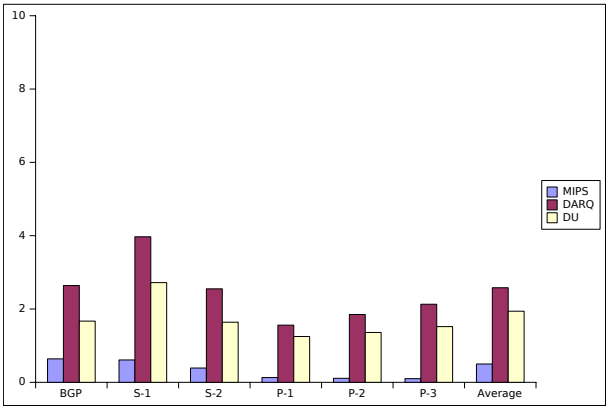
Table 3: Effect of compression ratio

5.2.2 Recall and Runtime

Table 4 shows the recall achieved by MIPS and DARQ in relation to the number of slices taken into consideration.



(a) RankMSE for query-wise selection



(b) RankMSE for subquery-wise selection

Figure 5: Mean squared errors with respect to ranking

Dataset	BGP	S-1	S-2	P-1	P-2	P-3	Total
Diseasome	7	21	21	8	20	12	89
Geo	10	15	23	0	0	0	48
LinkedMDB	5	0	0	0	0	0	5
Publication	1	2	5	0	0	0	8
Total	23	38	49	8	20	12	150

Table 5: Distribution of sub-query escaped by our approach

Our results display the superiority of our ranking function, which is always able to achieve a recall of 1 by querying 9 from 10 data sources. In contrast, DARQ always has to visit all data sources to achieve a comparable recall. We measured the number of sub-query executions that could be escaped by our approach without any loss in recall. The results shown in Table 5 clearly display that our approach can effectively reduce the total number of queries executed overall. Especially, on BGP queries, we were able to escape up to 10 subqueries. Our approach was most successful on star-shaped queries, where up to 21 subqueries could be escaped. The better ranking function also leads to a superior overall runtime as shown in Figure 6. In average, our approach requires solely between 79.85% (S-2 queries) and 93.46% (P-1 queries) of the runtime of DARQ to retrieve the complete results to each query.

6. DISCUSSION

7. REFERENCES

[1] Stuckenschmidt, H., Vdovjak, R., Houben, G.J., Broekstra, J.: Index structures and algorithms for querying distributed rdf repositories. In: WWW'04. (2004).

[2] Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. In: European semantic web conference on the semantic web: research and applications (ESWC 2008). LNCS, vol. 5021, pp. 524-538. Springer, Heidelberg (2008).

[3] Langegger, A., Waoay, W., Blaochl, M.: A semantic web middleware for virtual data integration on the Web. In: European semantic web conference on the semantic web: research and applications (ESWC 2008). LNCS, vol. 5021, pp. 493-507. Springer, Heidelberg (2008).

[4] Hartig, C. O, Bizer, J.-C. Freytag. Executing SPARQL queries over the web of linked data. 8th International Semantic Web Conference (ISWC 2009).

[5] Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U. Umbrich, J.: Data summaries for on-demand queries over linked data. In: (WWW 2010), USA.

[6] Umbrich, J., Hose, K., Karnstedt, M., Harth, A., Polleres.: Comparing data summaries for processing live queries over Linked Data. In:(WWWJ 2011), special issue querying the data web.

[7] Schwarte, .A, Haase, .P, Hose, .K, Schenkel, R, Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: 10th International semantic web conference (ISWC 2011).

[8] Perez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. In: 4th International

Rank	BGP		S1		S2		P1		P2		P3	
	MIPS	DARQ	MIPS	DARQ	MIPS	DARQ	MIPS	DARQ	MIPS	DARQ	MIPS	DARQ
1	29,22	19,22	26,53	24,42	30,69	30,69	18,12	15,67	24,60	21,99	31,60	29,30
2	49,61	39,50	46,20	39,67	52,15	49,70	34,04	31,49	38,84	38,56	43,78	45,02
3	67,48	61,78	60,90	54,78	66,48	59,96	50,28	49,23	55,74	53,16	61,86	56,76
4	78,44	72,68	70,75	63,86	76,60	71,49	65,71	64,04	72,25	69,11	80,17	73,72
5	85,65	79,26	79,76	74,32	83,64	80,54	77,24	79,87	83,04	86,11	90,45	91,74
6	90,54	84,00	86,67	79,59	89,90	86,76	82,28	83,75	87,69	89,95	94,48	95,70
7	94,68	87,25	92,94	85,61	94,88	90,22	85,56	87,17	89,99	92,08	95,59	96,73
8	99,40	89,38	98,62	89,98	99,35	93,43	87,71	89,29	91,47	93,31	95,70	97,53
9	100,00	97,95	100,00	96,52	100,00	97,06	100,00	94,97	100,00	97,15	100,00	99,26
10	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00	100,00

Table 4: Comparison of recall on top-k data sources

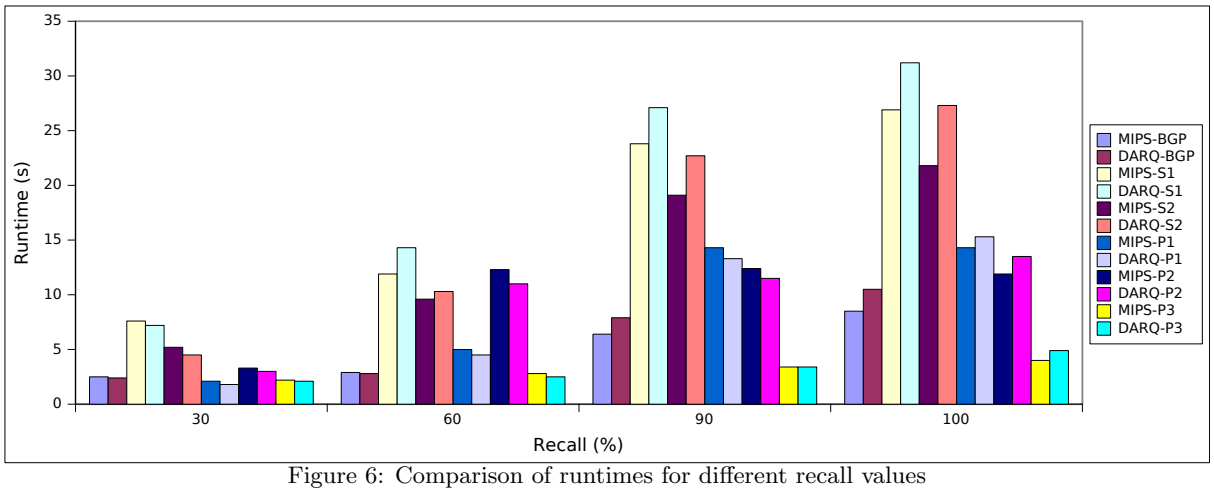


Figure 6: Comparison of runtimes for different recall values

SemanticWeb Conference (ISWC), Athens, GA, USA (November 2006).

[9] Broder, A. Z., Charikar, M., Frieze, A. M., & Mitzenmacher, M. (2000). Min-wise independent permutations. Journal of Computer and System Sciences, 60(3), 630-659.

[10] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. Communications of ACM, 13(7), 422-426.

[11] Mitzenmacher, M. (2002). Compressed bloom filters. In IEEE/ACM Transactions on Networking, 10(5), 604-612.

[12] Bender, M., Michel, S., Weikum, G., & Zimmer, C. (2005). Minerva: Collaborative p2p search. In Proceedings of the (VLDB) conference (Demonstration).

[13] Parreira, J.X, Michel, S., Weikum, G.: p2pDating: Real life inspired semantic overlay networks for Web search. In: Journal of Information Processing and Management 43 (2007) 643-664.

[14] http://www4.wiwiss.fu-berlin.de/lodcloud/state/

[15] Y. Li and J. Heflin. Using reformulation trees to optimize queries over distributed heterogeneous sources. In ISWC, pages 502-517, 2010.

[16] Z. Kaoudi, K. Kyzirakos, and M. Koubarakis. Sparql query optimization on top of dhts. In ISWC, pages 418-435, 2010.

[17] A. Deshpande, Z. G. Ives, and V. Raman. Adaptive query processing. Foundations and Trends in Databases, 1(1):1-140, 2007.

[18] G. Ladwig and T. Tran. Linked data query processing strategies. In ISWC, pages 453-469, 2010.

[19] G. Ladwig and T. Tran. Sihjoin: Querying remote and local linked data. In ESWC (1), pages 139-153, 2011.

[20] Auer.S, Lehmann. J, Ngonga, A.C.: Introduction to Linked Data and Its Lifecycle on the Web. In: Reasoning Web. Semantic Technologies for the Web of Data, Pages 1-75, 2011.

[21] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for sparql endpoints. In Proc. of International Semantic Web Conference (ISWC), 2011.

[22] C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and optimization of the sparql 1.1 federation extension. In ESWC (2), pages 1-15, 2011.

[23] . M. Stoker, A. Seaborne, A. Bernstein, C. Keifer, and D. Reynolds. SPARQL Basic Graph Pattern Optimizatin Using Selectivity Estimation. In WWW, 2008.

[24] C. Basca and A. Bernstein. Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In The 6th International Workshop on SSWS at ISWC, 2010.

[25] O. Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In ESWC, pages 154-169, 2011.

[26] Axel-Cyrille Ngonga Ngomo and Soren Auer. LIMES -

A Time-Ecient Approach for Large-Scale Link Discovery on the Web of Data. In Proceedings of IJCAI, 2011.

[27] R. Isele, A. Jentzsch, and C. Bizer. Ecient Multidimensional Blocking for Link Discovery without losing Recall. In WebDB, 2011.

[28] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. EAGLE: Ecient Active Learning of Link Specications using Genetic Programming. In: ESWC(2012).

[29] Unger, C.; Buhmann, L.; Lehmann, J.; Ngonga Ngomo, A.-C.; Gerber, D.; and Cimiano, P. Templatebased question answering over RDF data. In WWW (2012).

[30] G.Koutrika, and Y.Ioannidis, Personalized Queries under a Generalized Preference Model, In ICDE 2005.

[31] G.Koutrika, and Y.Ioannidis, Personalization of Queries in Database Systems, In ICDE 2004.

[32] S.Magliacane, A. Bozzon.E.D. Valle. Efficient Execution of Top-k SPARQL Queries. In: ISWC(2012), USA.

[33] B. J. Frey and D. Dueck. Clustering by passing messages between data points. Science, 315:972-976, 2007.

[34] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet Physics Doklady, volume 10, 1966.

[35] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL Benchmark - Performance Assessment with Real Queries on Real Data. In 10th Interna- tional Semantic Web Conference (ISWC2011), 2011.

[36] Ntarmos, N. and Triantafillou, P. and Weikum, G. Distributed hash sketches: Scalable, efficient, and accurate cardinality estimation for distributed multisets. ACM Trans. Comput. Syst. January 2009,2:1–2:53.

[37] Polyzotis, Neoklis and Garofalakis, Minos. Statistical synopses for graph-structured XML databases. Proceedings of the 2002 ACM SIGMOD international conference on Management of data. 2002,358–369.

[38] Natasha Drukh and Neoklis Polyzotis and Yossi Matias. Fractional XSKETCH synopses for XML databases. In Second International XML Database Symposium, XSym 2004, 189–203.

[39] Sebastian Michel, Matthias Bender, Peter Triantafillou,Gerhard Weikum. IQN Routing: Integrating Quality and Novelty in P2P Querying and Ranking. In Advances in Database Technology - EDBT 2006, 149-166.

[40] Moses S. Charikar.Similarity Estimation Techniques from Rounding Algorithms. Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, 380–388.