Multi-objective Linked Data Query Optimization

Technical Report

Günter Ladwig   Thanh Tran

Institute AIFB, Karlsruhe Institute of Technology, Germany

{guenter.ladwig,ducthanh.tran}@kit.edu

With the rapid proliferation of Linked Data on the Web, the topic of *Linked Data query processing* has recently gained attention. Works in this direction do not assume full availability of SPARQL endpoints. As an alternative to federation over these endpoints, this new querying paradigm follows the Linked Data principles to use only *HTTP lookups* for accessing and querying Linked Data. Existing works focus on the ranking and pruning of sources behind the query URIs, or on the efficient processing of data after they have been retrieved from sources. However, there exists no *systematic approach for query plan optimization*, especially the kind that considers both of these problems in a holistic way. Further, observing that result completeness is no longer a strict requirement and that there is an inherent trade-off between completeness, execution cost and other criteria, we propose a *multi-objective optimization* framework. In experiments on real world Linked Data, *Pareto-optimal plans* computed by our approach show benefits over suboptimal plans generated by existing solutions.

1 Introduction

In recent years, the amount of Linked Data on the Web has been increasing rapidly. SPARQL endpoints providing structured querying capabilities are available for some Linked Data sources such that federated query processing is possible. However, not all Linked Data providers, such as Website owners, can and want to serve their data through these endpoints. Instead, they follow the Linked Data principles [1], which dictate how to publish and to access Linked Data on the Web. According to these, dereferencing a Linked Data URI via HTTP should return a machine-readable description of the entity identified by the URI. As a result, while all Linked Data sources are accessible through HTTP lookups, only a few of them can be retrieved via SPARQL queries.

As an alternative to federation over remote SPARQL endpoints, researchers recently started to study the problem of Linked Data query processing [5, 3, 8, 4, 9], which relies only on HTTP lookups. One may argue that when remote SPARQL endpoints are not reliable or do not exist, Linked Data can be crawled and managed locally. Also with respect to this solution, Linked Data query processing can be seen as an alternative that provides up-to-date data and can be used in an ad-hoc fashion in scenarios where crawling and preprocessing are not affordable. In fact, it can also be seen as a focused crawling mechanism that can be run in batch mode for retrieving data for specific queries.

Src. 1: http://example.org/beatles

```
ex:beatles
  foaf:name "The␣Beatles" ;;
  ex:album ex:help ;
  ex:album ex:sgt_pepper .;
```

Src. 2: http://example.org/help

```
ex:help
  foaf:name "Help!"
  ex:song ex:night
  ex:song ex:girl .
```

Src. 3: http://example.org/sgt_pepper

```
ex:sgt_pepper
  foaf:name "Sgt.␣Pepper" ;
  ex:song ex:lucy ;
  ex:song ex:friends .
```

Src. 4: http://example.org/girl

```
ex:girl
  foaf:name "Another␣Girl" .
```

Src. 5: http://example.org/night

```
ex:night
  foaf:name "The␣Night..." .
```

Src. 6: http://example.org/lucy

```
ex:lucy
  foaf:name "Lucy..." .
```

Figure 1: Example Linked Data sources. The prefix *ex:* expands to "http://example.org/".

Example 1. *Figs. 1 & 2 show examples of Linked Data sources with their data and a SPARQL query, respectively. For processing such a query, a Linked Data query engine dereferences the URIs* ex:beatles, ex:sgt_pepper *and* ex:lucy *and retrieves the whole content of the sources 1, 3 and 6, respectively, via HTTP URI lookups. Then, links from these sources to other sources are explored for retrieving additional data that may be relevant, e.g. to follow the URI* ex:help *in source 1 to obtain source 2 (again, via HTTP lookup). While the data is being retrieved, the structured query is processed, i.e. matching triples from the incoming data streams are obtained for every triple pattern, which are then joined to produce* "Lucy..." *as a match for ?name.*

Clearly, the problems with this querying paradigm are (1) *expensive data access* through live lookups on the Web, (2) *large amount of data* to be processed because Linked Data sources have to be retrieved as a whole and a (3) *large number of sources* that have to be considered.

Existing works focus on the ranking and pruning of sources [8, 3], or on the efficient processing of data while it is retrieved from sources [5, 9], i.e. joins and traversal algorithms for retrieving and processing data from sources. However, there exists no *systematic approach for query plan optimization*, especially the kind that considers both the problems of source selection and data processing in a holistic way. Further, we observe that due to long execution times resulting from the large number of sources and their

```
1 SELECT ?name WHERE {
2    ex:beatles ex:album   ?album .
3    ?album     ex:song    ?song  .
4    ?song      foaf:name ?name  .
5 }
```

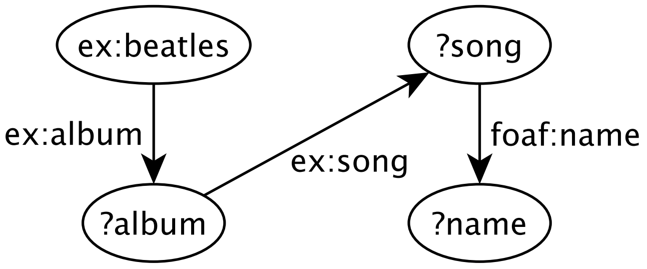


Figure 2: Example query asking for names of songs in albums of the Beatles. The query consists of three triple patterns t_1 (line 2), t_2 (line 3), and t_3 (line 4).

high processing cost, result completeness is often no longer affordable. Instead of assuming completeness and optimizing exclusively for cost, other criteria such as relevance, quality and cardinality of results, and trustworthiness of sources may be considered. This is problematic because these criteria are not always complementary. For instance, there is an inherent trade-off between output cardinality and cost: to produce more results, we have to retrieve more sources, which in turn increases processing cost. Taking this trade-off into account, we propose a *multi-objective optimization* framework. The contributions of this work can be summarized as follows:

(1) Solutions related to Linked Data query processing include (a) query processing strategies [5, 8] and (b) using Linked Data summaries [3] and heuristics for adaptive ranking to select only the few best sources [8]. However, there exists no works that systematically study the problem of query optimization, i.e. consider different query processing strategies as query plans, establish optimality criteria for these plans, and find the optimal ones. We propose the first solution *towards a systematic optimization of Linked Data query processing* that holistically considers these two (a) and (b) parts of the problem.

(2) In particular, we propose an optimization framework for Linked Data query processing, which incorporates *both standard query operators* and *source selection*. That is, we propose to extend the scope of query optimization from how to process data to also include which data to process. This is to reflect the nature of Linked Data query processing, where source selection and scanning become an essential part. Further, this framework supports the *joint optimization of several objectives*, cost and output cardinality in particular.

(3) We propose a *dynamic programming (DP) solution* for the multi-objective optimization of this integrated process of source selection and query processing. It produces a set of Pareto-optimal query plans, which represent different trade-offs between optimization objectives. The challenge of using DP here is that after retrieval, sources can be re-used in different parts of the query, i.e. the source scan operators can be shared. Depending on the reusability of these operators, the cost of subplans may vary such that

the cost function is no longer monotonic with regard to the combination of subplans. We provide a tight-bound solution, which takes this effect into account.

Outline. In Section 2 we introduce the problem of multi-objective Linked Data query processing. In Section 3 we propose an optimization framework for Linked Data query processing. Section 4 describes our adaptation of DP for multi-objective query optimization. Section 5 gives and overview of related work. Finally, we evaluate our approach in Section 6 before concluding in Section 7.

2 Problem Definition

We first define the data and query model and then introduce the problem of processing queries over Linked Data.

2.1 Linked Data and Queries

Linked Data on the Web today is basically RDF data managed according to the Linked Data principles. RDF is a graph-structured data model, where basic elements (called RDF terms) are URI references $\mathcal{U}$, blank nodes $\mathcal{B}$ and literals $\mathcal{L}$ that can be used to form triples, $\langle s,p,o\rangle \in (\mathcal{U}\cup\mathcal{B})\times\mathcal{U}\times(\mathcal{U}\cup\mathcal{B}\cup\mathcal{L})$, which together form an *RDF graph*. The Linked Data principles [1] mandate that (1) HTTP URIs shall be used and that (2) dereferencing such an URI returns a description of the resource identified by the URI, i.e. a set of triples where the URI appears as subject or object. Therefore, *an HTTP URI reference is treated as a data source*, called a *Linked Data source*, whose constituent triples can be retrieved by performing a HTTP URI lookup, and they contain other URI references leading to other sources (obtained via further URI lookups):

Definition 1 (Linked Data Source / Graph)**.** *A* Linked Data source*, identified by an HTTP URI d, is a set of RDF triples $\langle s,p,o\rangle$. There is a* link *between two Linked Data sources d_i, d_j if d_j appears as the subject or object in at least one triple of d_i, i.e. $\exists t \in T^{d_i}, t=\langle d_j,p,o\rangle \vee t=\langle s,p,d_j\rangle$ or vice versa, $\exists t\in T^{d_j}, t=\langle d_i,p,o\rangle\vee t=\langle s,p,d_i\rangle$. The union set of sources $d_i\in D$ constitutes the* Linked Data graph $T^D=\{t|t\in T^{d_i}, d_i\in D\}$*, where D denotes the set of all Linked Data sources.*

The standard for querying RDF data is SPARQL, of which basic graph patterns (BGP) are an important part. As illustrated in Fig. 2, a BGP is a set of triple patterns $Q=\{t_1,\dots,t_i,\dots,t_n\}$, $t_i=\langle s,p,o\rangle$, with s, p and o being either a RDF term (called *constant*) or a *variable*. Typically, every triple pattern in Q shares one common variable with at least one other pattern in Q such that Q forms a connected graph. Computing answers to a BGP query over the Linked Data graph amounts to the standard task of *graph pattern matching*, where a query result is a mapping, $\mu_{T^D}: V\rightarrow TERM$, from variables in the query graph Q to RDF terms in the Linked Data graph T^D. For intuitive presentation, we use the notation μ_{T^D} both to refer to the result nodes in T^D that match the query variables and the triples (subgraphs) that match a triple pattern (a BGP).

2.2 Processing Linked Data Queries

A BGP query is evaluated by first obtaining bindings for each of its constituent triple patterns $q \in Q$ and then performing a series of joins between the bindings. This is done for every two patterns that share a variable, forming a *join pattern* (that variable is referred to as the *join variable*). In the Linked Data context, BGP queries are not evaluated on a single source, but, in order to obtain all results, they have to be matched against the combined Linked Data graph T^D, where relevant sources in T^D have to be retrieved on the fly.

Previous work proposes exploration-based *link traversal* [5, 4] for obtaining relevant sources. These approaches take advantage of links between sources and discover new sources at run-time by traversing these links. For this, the query is assumed to contain at least one constant that is a URI. This URI is used for retrieving the first source, representing the "entry point" to the Linked Data graph. Triples in this entry point represent links to other sources. By following these links, new sources are discovered and retrieved. When retrieved sources contain data matching the query triple patterns, they are selected and joined to produce query results.

Given the large number of Linked Data sources and their high retrieval costs, it is often not practical to process all relevant sources. Thus, existing work does not guarantee *result completeness* but instead, ranks and processes the few best sources [3, 8]. The on-the-fly source exploration mentioned above has been combined with compile-time [3] and adaptive ranking of sources [8]. The idea is that, whenever statistics about sources are available, they can be exploited to find sources more effectively than zero-knowledge on-the-fly exploration. The most common statistics used is a *source index*, which maps a triple pattern t to URIs representing sources that contain results for t, i.e. $source(t) = \{d | d \in D \wedge, \mu_{T^d}(t) \neq \emptyset\}$. Often, the source index used by existing work not only returns the URIs but also selectivity information for triple and join patterns. These statistics are collected from previously explored sources or catalogs such as CKAN[1].

These existing works address the subproblems of (a) how to process SPARQL BGP queries in the Linked Data setting [5, 4, 8], i.e. when only URI lookups are available, and (b) how to obtain Linked Data summaries and statistics to select relevant sources [3, 8]. We focus on the *compile-time optimization of Linked Data query processing*, given the statistics acquired and stored in the source index. Compared to existing works above, the novelties are:

- We provide a framework for Linked Data query optimization: we show that Linked Data query processing can be formulated as query plans. Based on optimality criteria defined and metrics obtained for these plans, we systematically study the finding of optimal ones as an optimization problem.
- We propose a holistic solution to optimization that considers both the subproblems of (a) and (b), i.e. selecting and retrieving data from the sources as well as matching the query against this data to produce results. Further, this optimization is performed w.r.t. multiple objectives, which is solved through a novel DP solution. The optimization performed is done at compile-time, which as future work, can be

[1]`http://ckan.net`

integrated with adaptive optimization [8] that may also considers the exploration and retrieval of additional sources and computing their statistics on-the-fly.

3 Optimization Framework

The framework presented here provides the foundation for systematic solutions towards Linked Data query optimization, which consider the effect of query operators to compute and guarantee the (Pareto-)optimality of query plans.

3.1 Query Operators

The main difference to traditional federated query processing is that while (some) relevant sources can be determined using statistics in the source index, their entire content have to be retrieved via URI lookups as opposed to retrieving only the parts matching the query pattern by leveraging the structured querying capabilities of SPARQL endpoints. This is similar to a *table scan*. The difference is that sources have to be retrieved from remote sites. Moreover, several sources may contain answers for one single query predicate (triple pattern), and vice versa, one single source may be used for several predicates. We introduce the *source scan* operator to capture this:

Definition 2 (Source Scan)**.** *The input of a source scan operator $scan_d$ is the source URI d. Executing this operator outputs all triples in d, i.e. $scan_d = T^d$.*

Once source data is available, the query is processed using standard operators, i.e. operators also used for federated query processing. A selection $\sigma_{T^d}(t)$ is performed on the input T^d to output triples in T^d that match the triple pattern t. The outputs of two patterns t_i and t_j that share a common variable are joined as $t_i \bowtie t_j$. The union operator is used to combine results from different patterns, or from different sources for one pattern:

Definition 3 (Source Union)**.** $\bigcup(I_1, \ldots, I_n)$ *outputs the union of its inputs I_i, $1 \leq i \leq n$, where every input I_i may stand for results for one triple pattern, $I_i = \mu(t)$, or subexpression, e.g. $I_i = \mu(T)$. Because a triple pattern can match several sources, I_i may also capture partial results for a pattern t such that the union $\bigcup(\sigma_{T^{d_1}}(t), \ldots, \sigma_{T^{d_n}}(t))$ combines results from several selection operators.*

3.2 Query Plan

Query plans for relational databases generally consist of access plans for individual relations whose outputs are then processed by join and other operators. Here, we create an access plan for every triple pattern:

Definition 4 (Access Plan)**.** *Given a query Q, let $t \in Q$ be a triple pattern in Q and $D = source(t)$ be the set of sources for t. An* access plan *$p(t)$ for t is a tree-structured query plan constructed in the following way: (1) at the lowest level, the leaf nodes of $p(t)$ are source scan operators, one $scan_{d_i} = T^{d_i}$ for each $d_i \in D$; (2) the next level contains* *selection operators, one for processing the output of every scan operator, i.e. we have $\sigma_{T^{d_i}}(t)$ for every $d_i \in D$; (3) the root node is a union operator $\bigcup_t(\sigma_{T^{d_1}}(t), \ldots, \sigma_{T^{d_{|D|}}}(t))$ that combines the outputs of all selection operators for t.*

At the next levels, the outputs of access plans' root operators are successively joined to process all triple patterns of the query, resulting in a *tree of operators*. However, in Linked Data query processing, it is often the case that a single Linked Data source contains data matching several triple patterns. It is therefore possible that a data source is used for more than one query triple pattern. In this case it is detrimental to execute the scan operator more than once as this will incur network costs that can be avoided. We therefore perform *operator sharing*, where the output of a source scan operator is used as input for more than one selection operator, i.e. the output is shared. This means that access plans may overlap and the query plan is no longer a tree, but has the form of a directed acyclic graph (DAG) [11].

To avoid reading the outputs of shared operators multiple times, temporary buffers (on disk or in memory) can be used to store them, from which subsequent reads are served. This strategy however incurs overhead for writing and storing the temporary buffer. Thus, we propose the use of *push-based execution*, where (shared) operators simply push their outputs to all subsequent operators in the DAG, resulting in a reversed control flow compared to the classic iterator model. That is, instead of having several operators pulling and consuming their shared input independently, the output of a shared operator is pushed to and processed by all its consumers at the same time.

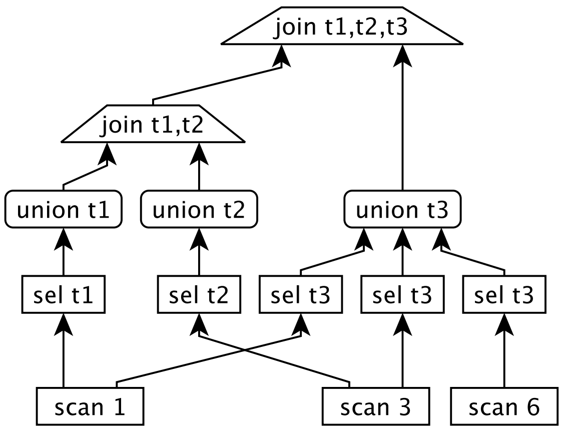


Figure 3: Query plan for the example query.

Example 2. *Assuming the source index returns that the relevant sources for the given query from Fig. 2 are* (1) ex:beatles*,* (3) ex:sgt_pepper *and* (6) ex:lucy*, Fig. 3 shows a corresponding query plan. To capture the sources, there are three corresponding source scan operators. Together with selection and union operators, the source scans form 3 access plans for the 3 triple patterns t_1, t_2, and t_3. The outputs of the access plans are then combined using 2 join operators.*

As the sources ex:beatles *and* ex:sgt_pepper *contain relevant data for more than one* *triple pattern (t_1, t_3 and t_2, t_3, respectively), their source scans are shared, i.e. the outputs of the these operators feed into 2 access plans.*

3.3 Pareto Optimal Plans

In standard cost-based optimization, completeness is assumed such that all results have to be computed. Optimality in this case is defined with respect to processing cost, and the goal is to find plans that are cost-optimal, i.e. to produce all results at lowest cost. Completeness is often not practical in Linked Data query processing and existing approaches select only a few best sources [3, 8] to terminate early. Not only cost but also the number of results and other aspects such as the trustworthiness of sources and the quality of data may play an important role. This is especially the case when Linked Data query processing is used in batch mode to crawl for data that meets certain criteria.

Multi-objective optimization can be used to support this. For a query Q, the goal is to compute the Pareto-optimal set of query plans that represents different trade-offs between multiple objectives. For clarity of presentation, we will focus on the two main objectives of maximizing output cardinality, $card(\cdot)$, and processing cost, $cost(\cdot)$. The Pareto-optimal set of solutions is defined using a *dominance* relation that incorporates the multiple objectives. A query plan is considered to dominate another plan, if it is at least as good in all objectives and better in at least one objective:

Definition 5 (Dominance)**.** *Given two query plans p_1 and p_2, p_1* dominates *p_2 ($p_1 > p_2$) if both the cost and cardinality of p_1 are "better" or equal to the cost and cardinality of p_2, and either the cost or cardinality is strictly "better" than the cost or cardinality of p_2, i.e. $cost(p_1) \leq cost(p_2) \wedge card(p_1) \geq card(p_2) \wedge ((cost(p_1) < score(p_2) \vee card(p_1) > card(p_2)) \Rightarrow p_1 > p_2$.*

Definition 6 (Pareto Optimal Plans)**.** *Given a query Q and a set of query plans $P(Q)$ for Q, the* Pareto-optimal *set $P^*(Q) \subseteq P(Q)$ comprises all plans that are not dominated by any other plan in $P(Q)$, i.e. $P^*(Q) = \{p_i \in P(Q) | \neg \exists p_j \in P(Q), p_j > p_i\}$. We denote the set of dominated plans as $P^-(Q) = P(Q) \setminus P^*(Q)$.*

4 DP Solution

In this section we propose a DP solution to the multi-objective Linked Data query optimization problem based on the original DP algorithm [17]. The original DP algorithm for query optimization works in a bottom-up fashion, constructing the query plan from the leaves, which are table scan operators to access relations. DP is used to deal with the exponentially large search space of possible query plans. It takes advantage of the *optimal substructure* of the problem, i.e. the optimal plan can be constructed from optimal subplans such that non-optimal subplans can be discarded during the process to reduce the search space.

For optimizing Linked Data query processing, we propose to construct access plans $P(t)$ for every triple pattern $t \in Q$. These *atomic plans* are then successively combined using join operators to create *composite plans* for larger subexpressions $T \subseteq Q$. For instance, to construct a query plan for the expression $T = t_1 \bowtie t_2$, the optimizer may consider all possible pairs $\{(p_1, p_2)|p_1 \in P(t_1), p_2 \in P(t_2)\}$ as possible combinations of plans. When combining two plans p_1, p_2 to form a new plan p, we write $p = \texttt{cmb}(p_1, p_2)$. At each stage, the optimizer reduces candidate subplans by discarding those that cannot be part of an optimal solution. That is, before constructing plans for larger subexpressions the optimizer creates $P^*(T) \subseteq P(T)$ for every subexpression T.

In the following, we firstly discuss how to use existing techniques to estimate the optimality of subplans for any expressions $T \subseteq Q$. We note that the focus of this work is not to obtain accurate cost and cardinality estimates but a DP solution that produces optimal plans by combining subplans (given their estimates). We discuss the main problems that arise when using DP for our problem. Firstly, we need to establish the *comparability of plans*, given there are multiple objectives. Further, because query plans are no longer required to produce all results, a *relaxation of the comparability constraint* is needed. Also, there is the effect of *operator sharing*. We will establish tight bounds on subplans' costs to deal with this effect and prove that the resulting multi-objective query optimization problem still has optimal substructure such that the proposed solution yields the Pareto-optimal solution.

4.1 Estimating Cost and Cardinality of Plans

We focus on basic estimates needed in this work that are commonly used and refer the readers to works on SPARQL selectivity estimation for more advanced techniques [12, 6].

Operators. The output cardinality of the source scan operator is the same as the size of the source, i.e. $card(scan_d) = |T^d|$. This source size statistics can be directly obtained from the source index. For union, cardinality is the sum of the cardinalities of its inputs: $card(\cup(I_1, ..., I_n)) = \sum_{i=1}^{n} card(I_i)$. The cardinality for selection and join depends on selectivity estimates $sel(\cdot)$, i.e. $card(\sigma_{T^d}(t)) = sel(t) \times |T^d|$ and $card(t_i \bowtie t_j) = sel(t_i \bowtie t_j) \times card(t_i) \times card(t_j)$, respectively. Costs for scan, selection, union and join are $cost(scan_d) = h_s \times |T^d|$, $cost(\sigma_{T^d}(t)) = h_\sigma \times |T^d|$, $cost(\cup) = h_\cup \times card(\cup)$, and $cost(\bowtie) = h_\bowtie \times card(\bowtie)$, respectively. This is because cost is typically assumed to be proportional to cardinality. The different weights $h_s, h_\sigma, h_\cup$, and $h_\bowtie$ can be used to reflect the ratio between the two. Typically, these parameters are tuned based on performance results observed from previous workloads and the availability of indexes and algorithms. For instance, $h_\bowtie$ depends on the join algorithm employed. In case of operator sharing, separate cost models for the first source scan (when the data is retrieved over the network) and subsequent scans (when the data has already been retrieved) are used. We use $cost_2(scan_d) = (1 - b) \times cost_1(scan_d)$, where $cost_1$ denotes first time cost, $cost_2$ stands for cost for each subsequent scan, and b is a parameter to control the benefit achievable through operator sharing.

Atomic Plan. The cardinality of an access plan $p(t)$ is captured by its root node, i.e. $card(p(t)) = card(\cup_t)$. Its cost is calculated as the sum of the cost of its nodes. Source scan nodes are marked after first time usage so that the right cost model can be determined for this calculation.

Composite Plan. Composite plans capture the joins between results obtained for several triple patterns (outputs of access plans). For an expression $T = t_i \bowtie t_j$, $card(p(T)) = card(t_i \bowtie t_j)$ and $cost(p(T)) = cost(t_i \bowtie t_j)$.

4.2 Comparability

Comparability is defined as an equivalence relation $\sim$ over plans. It determines which plans are comparable, based on which the optimizer decides which plans are suboptimal and then prunes all but the optimal plans for each equivalence class induced by $\sim$.

In the traditional setting, atomic operators and plans are *comparable when they produce the same results.* This comparability relation is applicable there because input relations are fixed given the query such that operators used to process them produce the same output and vary only with regard to cost, i.e. plans are compared only w.r.t. cost because they produced the same results. The optimizer only chooses how to process data (e.g. table or index scan) based on cost estimates. In Linked Data query processing, however, the selection of sources (represented by source scan operators) is part of query optimization. Thus, the optimizer decides both what and how data shall be processed, i.e. plans have to be compared w.r.t. cost and the results they produce. If we apply the comparability concept as defined previously, each unique combination of source scan operators may yield different results and thus, constitutes a separate equivalence class of query plans. This limits the number of comparable plans and hence, those that can be pruned.

However, we note that given the objectives here are cardinality and cost, we are not interested in which results but how many results will be produced. Accordingly, a relaxation of this comparability relation can be employed that enables the optimizer to prune plans more aggressively.

Definition 7. *Two query plans p_i, p_j are* comparable *if they produce results for the same expression, i.e. $p_i(T_i) \sim p_j(T_j)$ if $T_i = T_j$.*

This relaxation means that plans can be compared even when they do not produce exactly the same results. The equivalence class of comparable plans is enlarged to include all plans that produce the same type of results (bindings for the same pattern). As a consequence, the query can be decomposed into subpatterns, and plans constructed for subpatterns can compared w.r.t. the objectives.

4.3 Monotonicity and Dominance

Every objective can be reflected by a scoring function. When combining plans for subpatterns to successively cover a larger part of the query, the scores of these subplans have to aggregated. For pruning suboptimal plans, a central requirement for the DP solution is that the scoring function must be *monotonic* with respect to plan combination. Only then, it can be guaranteed that some subplans can be safely pruned because they cannot be part of optimal plans. We now discuss monotonicity w.r.t. the scoring functions for the objectives of cost and cardinality, and show under which conditions pruning is possible.

Cardinality. Atomic plans are combined to capture joins between results. The monotonicity of the cardinality scoring function can be established because the cardinality function for join is monotonic:

Lemma 1. *Given a query Q, let $T, T' \subset Q$ be two subexpressions of Q, such that $T \cap T' = \emptyset$. Let $p_1, p_2 \in P(T)$ and $p' \in P(T')$ be plans for T and T'. Then we have $card(p_1) \leq card(p_2) \Rightarrow card(\texttt{cmb}(p_1, p')) \leq card(\texttt{cmb}(p_2, p'))$.*

Proof. The combination above captures the expression $T \bowtie T'$. Based on the definition of $card(T \bowtie T')$, we write the condition in the lemma as $card(p_1) \leq card(p_2) \Rightarrow card(p_1) \times card(p') \times sel(T \bowtie T') \leq card(p_2) \times card(p') \times sel(T \bowtie T')$. This is true due to monotonicity of multiplication. □

Cost. For cost estimation, operator sharing is taken into account. Because the costs of first and subsequent scans vary, the cost of the source scan operator changes when a plan is combined with another plan that shares that operator. Suppose we have two plans p, p' for the subexpression $T \subset Q$ and $cost(p) > cost(p')$, and a plan p_t for a triple pattern t such that $Q = T \cup t$. The optimizer would consider p' to be the optimal plan for T and discard p to form $P^*(T) = \{p'\}$. Now, due to operator sharing it is possible that the cost of the combination of two plans is less than the sum of the cost of the two combined plans, i.e. it is possible that $cost(\texttt{cmb}(p, p_t)) < cost(\texttt{cmb}(p', p_t))$ if p and p_t share the same source such that the cost of p_t when combined with p is much lower than the cost of p_t that is combined with p'. In this case, p' is not part of $P^*(T)$.

Cost Bounds for Partial Plans. In order to take this effect of operator sharing into account when calculating the cost of a partial plan p, we define upper and lower bounds for p based on larger plans that use p as subplans:

Definition 8 (Lower and Upper Bound Cost)**.** *Given a query Q, the subexpressions $T \subset Q$, $T' = Q \setminus T$, a plan $p \in P(T)$, and let $P^p(Q) \subseteq P(Q)$ be the set of all plans for Q that are constructed as combinations of p and plans in $P(T')$: $P^p(Q) = \{\texttt{cmb}(p, p') | p' \in P(T')\}$. Then, we have* lower bound cost *for p as $cost_L^Q(p) = MIN\{cost(\texttt{cmb}(p, p')) | \texttt{cmb}(p, p') \in P^p(Q)\}$ and* upper bound cost *for p as $cost_U^Q(p) = MAX\{cost(\texttt{cmb}(p, p')) | \texttt{cmb}(p, p') \in P^p(Q)\}$.*

Intuitively, a plan p_i for a subexpression T of Q is "worse" in terms of cost than another plan p_j for T, if all plans for Q that are based on p_i have higher cost than all plans for Q that are based on p_j, i.e., if $cost_L^Q(p_i) > cost_U^Q(p_j)$. Based on these bounds, we can establish the monotonicity of plan cost with respect to plan combination as follows:

Lemma 2. *Let $T, T' \subset Q$ be two subexpressions of Q such that $T \cap T' = \emptyset$, and $p_1, p_2 \in P(T)$ and $p' \in P(T')$ be plans for T and T', respectively. We have*

$$cost_U^Q(p_1) \leq cost_L^Q(p_2) \Rightarrow cost_U^Q(\texttt{cmb}(p_1, p')) \leq cost_L^Q(\texttt{cmb}(p_2, p'))$$

Proof. Any plan for Q that is constructed as the combination $p'_1 = \mathtt{cmb}(p_1, p')$, i.e., any plan in $P^{p'_1}(Q)$, is also a p_1-combination (because p'_1 is constructed based on p_1) such that $P^{p'_1}(Q) \subseteq P^{p_1}(Q)$ and thus, $cost_U^Q(p'_1) \leq cost_U^Q(p_1)$. Analogously, for p_2 and $p'_2 = \mathtt{cmb}(p_2, p')$, we have $cost_L^Q(p'_2) \geq cost_L^Q(p_2)$. Hence, $cost_U^Q(p_1) \leq cost_L^Q(p_2) \Rightarrow cost_U^Q(p'_1) \leq cost_L^Q(p'_2)$. □

Based on these results for cardinality and cost monotonicity, we now refine the dominance relation to make it applicable to subplans, i.e., plans for strict subexpressions of Q:

Theorem 1. *Given a query Q, a subexpression $T \subset Q$ and two comparable plans $p_1 \sim p_2$ for T, $p_1 > p_2$ if $card(p_1) \geq card(p_2) \wedge cost_U^Q(p_1) \leq cost_L^Q(p_2) \wedge (card(p_1) > card(p_2) \vee cost_U^Q(p_1) < cost_L^Q(p_2))$.*

This is the main result needed for pruning. A subplan is suboptimal and thus can be pruned if it is dominated in the sense specified above.

Cost Bound Estimation. A basic strategy to compute the lower and upper bounds of a plan p is to construct all plans based on p. This is of course very cost intensive and defeats the purpose of pruning. Observe that for pruning, we need only to compare the upper and lower bounds between pairs of plans p_1, p_2 for the subexpression $T \subset Q$. Given p_1, p_2 can be pruned if it has higher cost when used to process T, and further, when its *benefit* that may arise when processing other parts of the query cannot outweigh this difference in cost. If exists, this benefit can be completely attributed to operator sharing. Hence, for the efficient estimation of bounds, we propose to focus on the maximal benefit that is achievable through operator sharing. As the source scan is the only shareable operator, we derive the maximal benefit of one plan p_2 compared to another p_1 through a comparison of their source scan operators. In particular, only those source scans captured by p_2 not covered by p_1 (i.e. the additional benefit achievable with p_2) have to be considered:

Definition 9 (Maximal Benefit)**.** *Given a query Q and two query plans $p_1, p_2 \in P(T), T \subset Q$, let D_{p_1}, D_{p_2} be the sets of sources (respectively the source scan operators) used by p_1 and p_2, respectively, D'_{p_2} be the set of sources used by p_2 not covered by p_1, i.e. $D'_{p_2} = D_{p_2} \setminus D_{p_1}$ and Q' be the set of triple patterns not covered by p_1 and p_2, i.e. $Q' = Q \setminus T$, the* maximal benefit *of p_2 given p_1 is $mb(p_2|p_1) = \sum_{t \in Q'} \sum_{d \in source(t), d \in D'_{p_2}} b \cdot cost_1(scan_d)$.*

Lemma 3. *Given a query Q, a subexpression $T \subset Q$ and two plans p_1, p_2, if $cost(p_1) \leq cost(p_2) - mb(p_2|p_1)$ then $cost_U^Q(p_1) \leq cost_L^Q(p_2)$.*

Proof. As plans p_1, p_2 are both in $P(T)$ they both can be combined with the same set of plans for $P(Q \setminus T)$, meaning that the only difference in final plans built for p_1 and p_2 lies in the shared source scan operators. If we now know that p_1 has lower cost than p_2 even when the maximal benefit for p_2 obtainable from operator sharing is considered, then the upper bound cost $cost_U^Q(p_1)$ is also lower than $cost_L^Q(p_2)$. □

Based on these bounds defined w.r.t. the maximal benefit, we finally obtain the following dominance relation:

Theorem 2. *Given a query Q, a subexpression $T \subset Q$ and two plans $p_1 \sim p_2 \in P(T)$, $p_1 > p_2$ if $card(p_1) \geq card(p_2) \wedge cost(p_1) \leq cost(p_2) - mb(p_2|p_1) \wedge (card(p_1) > card(p_2) \vee cost(p_1) < cost(p_2) - mb(p_2|p_1))$.*

4.4 Pareto-optimality

The goal of the optimizer in Linked Data query processing is to find the Pareto-set of query plans, while pruning as many plans as possible at each step. We now show that pruning suboptimal plans based on the comparability and dominance relations established previously yields the complete Pareto set $P^*(Q)$, i.e., the proposed multi-objective optimization still has optimal substructure. Given the decomposition of Q into the subproblems $T \subset Q$, we construct $P^*(Q)$ as a combination of optimal subsolutions $P^*(T)$. This means a non-optimal solution for a subproblem T must not be part of an optimal solution for Q:

Theorem 3. *Given a query Q and two subexpressions $T_1, T_2 \subseteq Q$ with $T_1 \cap T_2 = \emptyset$, the set of optimal plans for $T_1 \cup T_2$ can be constructed from optimal plans for T_1, T_2, i.e., $P^*(T_1 \cup T_2) \subseteq \{\mathtt{cmb}(p_1, p_2) | p_1 \in P^*(T_1), p_2 \in P^*(T_2)\}$.*

Proof. We prove this by contradiction: Let $p^* \in P^*(T_1 \cup T_2)$ be a plan that is a combination of a dominated plan for T_1 and a non-dominated plan for T_2, i.e., $p^* = \mathtt{cmb}(p_1^-, p_2^*), p_1^- \in P^-(T_1), p_2^* \in P^*(T_2)$. This means, there must be a non-dominated plan $p_1^* \in P^*(T_1)$ that dominates p_1^-, but the combination of p_1^* with p_2^* is dominated by the combination of p_1^- and p_2^*:

$$\exists p_1^* \in P^*(T_1) : \mathtt{cmb}(p_1^-, p_2^*) \text{ dominates } \mathtt{cmb}(p_1^*, p_2^*)$$

Given p_1^* dominates p_1^- and $\mathtt{cmb}(p_1^-, p_2^*)$ dominates $\mathtt{cmb}(p_1^*, p_2^*)$, it follows from the established dominance relation that (without loss of generality, we use strictly lesser/greater relations):

$$card(p_1^-) < card(p_1^*) \wedge card(\mathtt{cmb}(p_1^-, p_2^*)) > card(\mathtt{cmb}(p_1^*, p_2^*))$$
$$cost_L^Q(p_1^-) > cost_U^Q(p_1^*) \wedge cost_U^Q(\mathtt{cmb}(p_1^-, p_2^*)) < cost_L^Q(\mathtt{cmb}(p_1^*, p_2^*))$$

However, this contradicts with the monotonicity property established for cost, because $cost_L^Q(p_1^-) > cost_U^Q(p_1^*)$, but $cost_U^Q(\mathtt{cmb}(p_1^-, p_2^*)) < cost_L^Q(\mathtt{cmb}(p_1^*, p_2^*))$. Analogously, a contradiction also follows from the monotonicity of cardinality. With regard to our original proposition, this means that there is no plan $p^* \in P^*(T_1 \cup T_2)$, such that p^* is a combination of a dominated plan p_1^- and a non-dominated plan p_2^*. This obviously also holds true when p^* is a combination of two dominated plans. Thus, all $p^* \in P^*(T_1 \cup T_2)$ must be combinations of non-dominated plans in $P^*(T_1)$ and $P^*(T_2)$ and therefore $P^*(T_1 \cup T_2)$. □

4.5 Optimizer Algorithm

In this section we present a DP algorithm that exploits the previously established theoretical results to perform multi-objective Linked Data query optimization. The proposed solution shown in Alg. 1 takes the proposed structure of Linked Data plans into account and uses Pareto-optimality to prune plans according to the optimization objectives.

```
Algorithm 1: PlanGen($Q$)
Input: Query $Q = \{t_1, \ldots, t_n\}$
Output: Pareto-optimal query plans $P^*(Q)$
1 foreach $t \in Q$ do
2     $S \leftarrow \{\cup(\{\sigma_{T^d}(t) | d \in D\}) | D \in \mathcal{P}(source(t))\}$
3     $P^*(t) \leftarrow \{p \in S | \nexists p' \in S : p' > p\}$
4 for $i \leftarrow 2$ to $|Q|$ do
5     foreach $T \subseteq Q$ such that $|T| = i$ do
6         foreach $t \in T$ do
7             $S \leftarrow S \cup \{\texttt{cmb}(p_1, p_2) | p_1 \in P^*(t), p_2 \in P^*(T \setminus t)\}$
8         $P^*(T) \leftarrow \{p \in S | \nexists p' \in S : p' > p\}$
9 return $P^*(Q)$
```

In the first step, access plans for single triple patterns are created (lines 1-3). For each triple pattern t in Q, relevant sources are determined using the source index. As we need to consider all possible combinations of sources, we create the power set $\mathcal{P}(source(t))$ of all sources (line 2). For each member D of the power set, we create an access plan, consisting of a scan and a selection operator $\sigma_{T^d}(t)$ for each source $d \in D$ and a single union operator $\cup$ that has the selection operators as input. S then contains a set of access plans, one for each combination of relevant sources. From this set of comparable plans (they cover the same pattern t), we then select only the non-dominated access plans and store them in $P^*(t)$ (line 3).

During the next iterations (line 4-8), previously created plans are combined until all query triple patterns are covered. For iteration i, we select all subsets $T \subseteq Q$ with $|T| = i$. For each $t \in T$ the algorithm creates all possible combinations between the Pareto-optimal plans for t and $T \setminus t$ (line 7). All these plans are stored in S. They are comparable since they cover the same triple patterns T. Finally, only the non-dominated plans from S are selected and stored in $P^*(T)$ (line 8). After the last iteration, $P^*(Q)$ contains all the Pareto-optimal plans for Q (line 9).

Complexity. The join order optimization problem has been shown to be NP-complete [19] and the classic DP algorithm for query optimization has a time complexity of $O(3^n)$ [7], where n is the number of relations (triple patterns in the case of Linked Data queries) to be joined. Our approach for multi-objective query optimization adds the dimension of source selection to the query optimization problem. Given a set of $|D|$ sources, we can think of the problem as, in worst case, creating a query plan for each unique combination of sources, of which there are $2^{|D|}$, leading to a complexity of $O(2^{|D|} \cdot 3^n)$. This theoretical worst case complexity does not change in the multi-objective case. However in practice, the number of plans that can be pruned at every iteration can be expected to be much larger in the single-objective case, compared to the multi-objective case. One strategy to deal with that is to approximate the bounds that we have established. In the experiment, we study one basic approximation, which instead of the cost bounds, use actual cost for pruning. That is, it ignores the bounds and accepts the discussed cases where subplans, which become non-optimal through operator sharing, may be part of the final result.

5 Related Work

We introduced the problem of *Linked Data query processing* and discussed existing solutions for that in Section 2.2. We showed that the proposed solution is the first systematic approach that solves the finding of plans as an optimization problem. In particular, it considers the entire querying process, from source selection to processing data retrieved from sources, as well as multiple objectives. Also, the differences to *federated query processing* have been discussed: there are no endpoints that can answer parts of the structured query such that the problem here is not the composition of views [15] or joined partial results retrieved from endpoints but the selection of sources and the processing of the entire sources' content. We will now discuss other directions of related work.

Source Selection. The problem of selecting relevant sources has been a topic in data integration research [10]. In this setting, sources are described not only by their content, but also their capabilities. Algorithms have been proposed to efficiently perform source selection by using the source characteristics to prune the search space. However, in these approaches, source selection here is a separate step that is decoupled from query optimization. In [13] the authors recognize that the decoupling of source selection and query optimization leads to overall sub-optimal plans and propose a solution that optimizes not only for cost but also coverage. A (weighted) utility function is proposed to combine them into a single measure. Finding the right utility function is generally known to be difficult, especially when many objectives have to be considered. Instead, we follow a different direction, employing multi-objective optimization to produce Pareto-optimal plans that represent different trade-offs between the objectives.

Query Optimization and Processing. There is a large amount of database research on query optimization. The DP solution was first proposed in [17]. Efficiently generating optimal DAG-shaped query plans when performing operator sharing has been addressed in [11]. In our work we also uses operator sharing for dealing with Linked Data sources. However, the effect of this is different in our multi-objective optimization problem, where we introduce special bounds needed for pruning. The efficient execution of DAG-shaped plans was discussed in [11], where several approaches were proposed, including the push-based execution that is used in our implementation.

Multi-objective Query Optimization. To the best of our knowledge, [14] is the only work addressing multi-objective query optimization, where it is studied in the context of Mariposa [18], a wide-area database. The optimizer splits the query tree into subqueries and then obtains bids from participating sites that specify a delay and cost for delivering the result of a subquery. The goal of the proposed multi-objective optimizer [14] is to obtain the Pareto optimal set of plans with respect to cost and delay. The problem studied there is different because there is only a single query operation tree and for each node, the optimizer has a list of alternatives for implementing the operation. In contrast, our solution does not consider a single query tree (i.e. a single order of operations) but all possible query trees to construct optimal plans in a bottom-up fashion.

Skyline Queries. The skyline operation finds non-dominated points from a set of points and has been used in conjunction with standard relational algebra [2] to compute results. The problem tackled by our approach is not computing results but query plans. As a result, the relaxed comparability, the conditions under which the scoring functions are monotonic, the estimation of bounds as well as the proposed DP algorithm are specific to our problem setting.

6 Evaluation

6.1 Systems

Our Approach. We implemented three versions of our approach. The first version (DP) implements all the proposed techniques. The second version (DPU) also uses operator sharing. However, it uses directly the cost instead of the lower and upper bounds that have been established to guarantee monotonicity of cost. While DPU might compromise Pareto-optimality (it is an approximate version of our DP solution), it can prune more aggressively and thus, is expected to exhibit better time performance than DP. The third version (DPS) does not use operator sharing. We use different settings for b to study the effect of operator sharing. For example, with $b = 0.8$ the optimizer assumes that 80% of the source scan cost is saved, i.e., subsequent reads cost only 20% of the first read.

Baselines. Existing Linked Data approaches implement source ranking to select few best sources [3, 8], and then process these sources without further optimization. This processing represents one single plan, whose optimality is unknown. We implement existing source ranking strategies [3, 8] (RK) and a random source selection strategy (RD). Then, given the selected sources, we apply our DP solution on top but only to optimize the cost. Thus, these baselines implements single-objective cost-based optimization where source selection and query processing is decoupled. Based on them, we study the effect of the holistic treatment of source selection and query processing and the multi-objective optimization performed by our approach.

Instead of one single cost-optimized plan that returns a certain number of results, our approach yields a Pareto-set of query plans with varying numbers of results. To produce different number of results, we extend these baselines to obtain several cost-optimized plans for different combinations of sources. Both baselines first retrieve all relevant sources D for a query Q from the source index, i.e. $D = \bigcup_{t \in Q} source(t)$. Then, a set $\mathcal{D}$ containing $|D|$ different subsets of D, each with size in the range $[1, |D|]$ is created. The baselines differ in how these subsets are selected.

- Baseline RD randomly selects the $|D|$ subsets.
- Baseline RK first ranks sources in D by the number of contained triples that match query triple patterns, calculated as $score(d) = \sum_{t \in q} card_d(t)$. The subsets are created by starting with the highest ranked source and then successively adding sources in the order of their rank to obtain $|D|$ subsets in total.

Each element in $\mathcal{D}$ represents a combination of sources. For each of them, a cost-optimized query plan is created. As a result, these baselines yield a set of plans, which vary in the number of results as well as cost.

Note that our approach not only selects sources (source scan operators) but also for which triple patterns these sources are used (selection operators), while sources selected for the baselines are used for all triple patterns. To obtain even more plans that further vary in the selection operators used, we create an additional set of m plans for each previously created plan of the baselines RK and RD by randomly removing a subset of the inputs (selection operators) from their access plans. In the end, each baseline has at most $m \cdot |D|$ plans. In this experiment, m is used as a parameter to control the number of plans generated by the baselines.

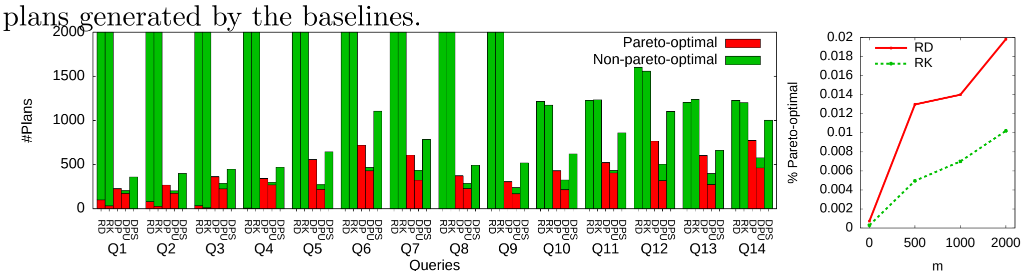


Figure 4: a) Number of Pareto-optimal and non-pareto-optimal plans for all queries and systems ($b = 0.8, m = 2000$), b) Pareto-optimal fraction for RD, RK for different value of m ($b = 0.8$).

6.2 Setting

Extending the Linked Data query set published in the recent benchmark [16], we obtain 14 BGP queries that have non-empty results. These queries belong to different classes of complexity, which is reflected in the number of triple patterns. For the classes of 3, 4 and 5 patterns, we have 4, 5, and 5 queries, respectively.

As data, we use real-world Linked Data on the Web. Processing the 14 queries against Linked Data sources on the Web involves a total of 1,909,109 triples from 516,293 sources capturing information from popular datasets such as DBpedia, Freebase, New York Times and LinkedMDB.

We observe that network latency varies. In order to establish a controlled environment and ensure the repeatability of the experiments, we simulate source loading to obtain a fixed delay of 1.5s that was observed to be representative of real Linked Data sources [8]. We also experimented with different delays, but performance differences between systems were however not sensitive to these settings.

All systems were implemented in Java. All experiments were executed on a system with a 2.4 GHz Intel Core 2 Duo processor, 4GB RAM (of which 1GB was assigned to the Java VM), and a Crucial m4 128GB SSD.

6.3 Results

Pareto-optimality. Fig. 4a displays the number of plans that were generated by each system, categorized into Pareto-optimal and non-pareto-optimal (i.e. dominated) plans. The Pareto-optimal plans were determined by collecting all plans from all systems and then pruning all dominated plans. We can see that DP produces only Pareto-optimal plans and that there are many DPU plans that are part of the Pareto-optimal set (56% on average). However, the RD and RK baselines generate only small fractions of Pareto-optimal plans (1.9% and 1% on average). Also, DPS finds only few Pareto-optimal plans (less the 1%).

Fig. 4b shows the Pareto-optimal fraction for RD and RK for different values of m. For larger values the Pareto-optimal fraction is higher, meaning that the larger plan space created by randomly removing source inputs is necessary to find Pareto-optimal plans.

Figs. 6a+b show plots of cost and cardinality of plans generated by all systems for query Q1. In these plots, a plan dominates all other plans that are to its lower right. We can see that many of the plans generated by the RD and RK baselines are dominated by other plans and that all DPS plans are also suboptimal. Fig. 6b shows for all systems only the plans that are part of the Pareto-optimal set. Here, the dominated DPS plans no longer appear and only few RD and RK plans remain.

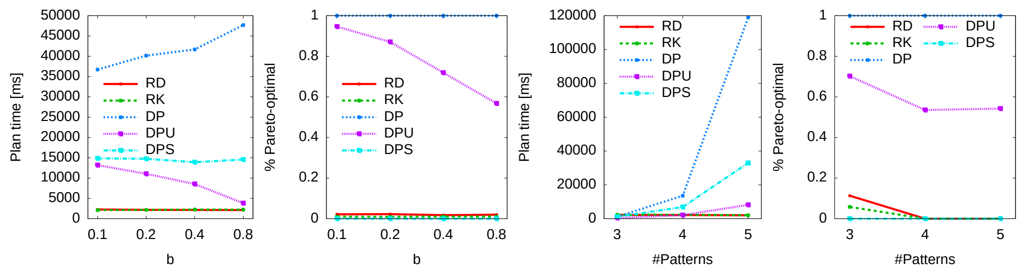


Figure 5: Effect of sharing benefit on a) planning time and b) Pareto-optimal fractions ($m = 2000$). Effect of query complexity on c) planning time and d) Pareto-optimal fractions ($b = 0.8, m = 2000$).

Thus, ranking sources based on cardinality only does not help to produce Pareto-optimal plans. Further, this bias towards cardinality as reflected by the RK baseline actually leads to a smaller amount of optimal plans, compared to RD, the random strategy (Fig. 4b). DPU optimizes for both objectives, thus is able to produce better trade-offs than RK and RD in most cases (Fig. 6a). However, because it only uses approximate estimates for cost, the resulting plans are relatively “good” but not always optimal.

Planning Time. On average, the fastest systems are RD and RK, while DP is more than one order of magnitude slower. This is to be expected because RD and RK randomly choose plans and use only simple source ranking, respectively, while DP requires computing precise bounds and finding Pareto-optimal plans using these bounds.

Interestingly, the approximate version of DP, DPU, can be as fast as RD and RK, and is only 3 times slower on average. DPU is not only faster than DP but also DPS. Differences between DPS, DPU and DP are due to operator sharing. DPS is faster than DP because without operator sharing, it saves time for computing bounds. However, because operator sharing results in greater cost differences between plans, DPU could prune more plans compared to DPS (while the overhead it incurs for bound estimation is small). This is more obvious when we vary the sharing benefit, as discussed in the following.

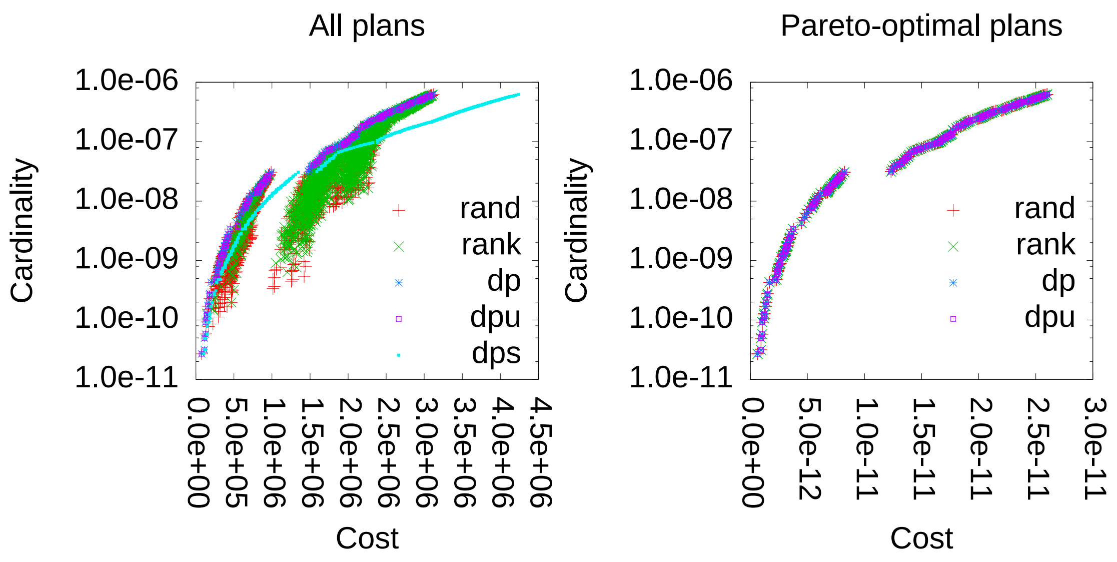


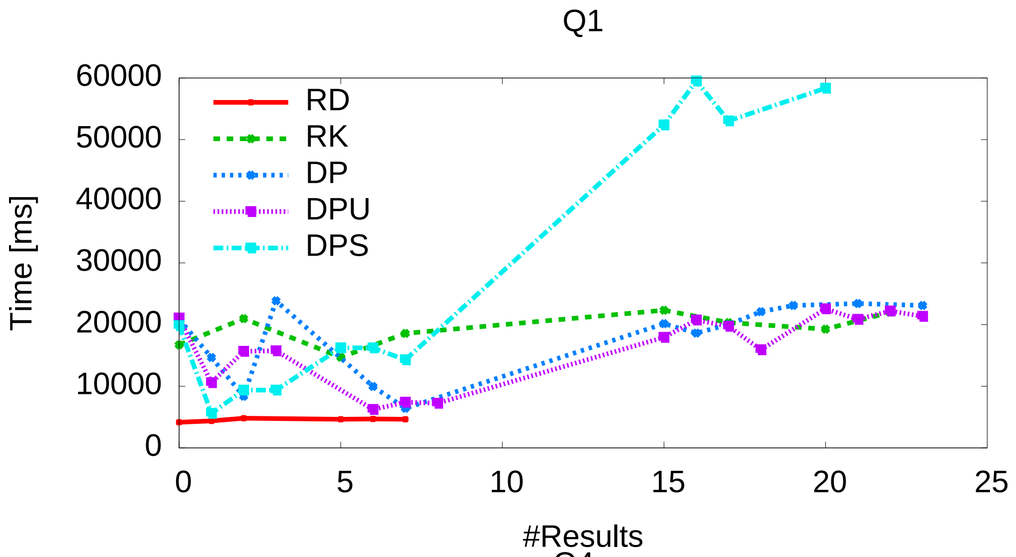
Figure 6: Plans for query Q1 on all systems: a) all plans and b) Pareto-optimal plans ($b = 0.8, m = 2000$).

Effect of Sharing Benefit. Figs. 5a+b show the planning time and Pareto-optimal fraction for different values of b. We see in Fig. 5a that planning times for systems without operator sharing (DPS, RD and RK) are not affected by b. For DP, planning time increases with higher sharing benefits, namely from 36.7s for $b = 0.1$ to 47.7s for $b = 0.8$. This is because cost bounds are more loose with increasing benefit, and thus less plans can be pruned. DPU's planning time exhibits the opposite behavior, decreasing from 13.2s ($b = 0.1$) to 3.8s ($b = 0.8$). Compared to DP, DPU does not incur the high cost of estimating bounds, and also, does not have the problem of loose bounds. Higher benefits only create steeper cost gradient between plans, thus resulting in more plans that can be pruned.

Not taking precise bounds into account however has a negative effect on the optimality of plans. Fig. 5b illustrates that DPU produces a smaller fraction of Pareto-optimal plans. This is because with higher sharing benefit, the deviation of DPU's bound estimates from the actual bounds increases.

In total, DPU however represents a reasonable trade-off between plan quality and time, producing 55 times more Parato-optimal plans while being only 3 times slower than the baselines RD and RK on average.

Effect of Query Complexity. Figs. 5c+d show time and Pareto-optimal fraction for different numbers of triple patterns. An increased number of patterns results in a larger search space for the query optimizer. As a result, both performance and quality decrease. Whereas for 3 triple patterns the baselines RD and RK are able to find 11% and 6% of the Pareto-optimal plans, few are found for 4 and 5 triple patterns (¡ 1%). DPU provides 70% of the Pareto-optimal set for 3 triple patterns, and 54% for 4 and 5 triple patterns. For all systems, planning time increases with the number of patterns. From 3 to 5 triple patterns, the planning time of DP increases by a factor of 101.4, DPS increases by a factor of 28, while the planning time for DPU only increases by a factor of 18, and RD and RK are largely unaffected.



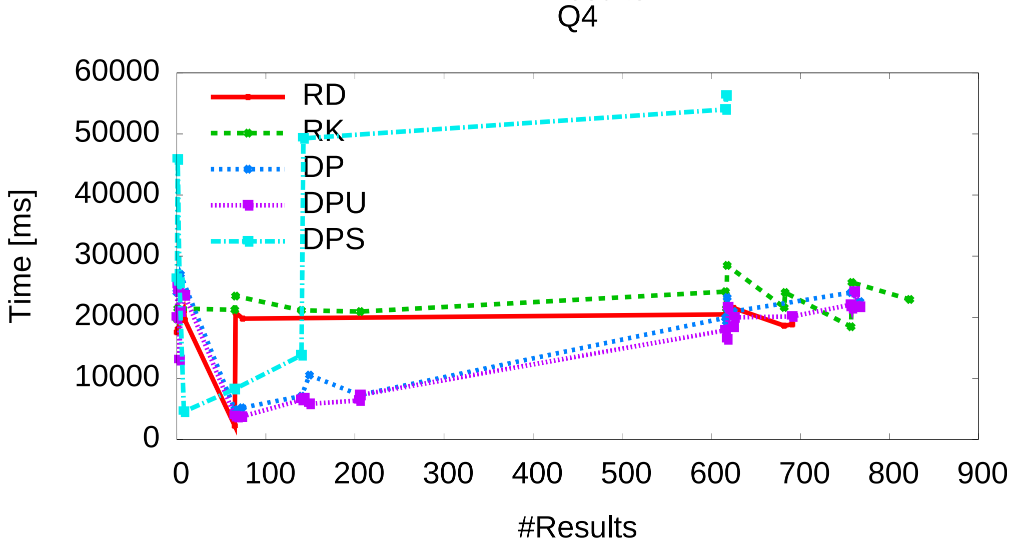


Figure 7: Execution times of query plans for queries Q1 and Q4.

Cost-Cardinality Trade-off. We analyze the cost-cardinality trade-off by studying the times needed for producing different number of results. Different number of results can be obtained by using different plans. For every query, we randomly chose 20% of the plans generated by each system, execute all of them and record the total time of planning and processing. Fig. 7 shows the results for two extreme queries. While Q1 produces only 24 results, Q4 yields 836 results. Each point represents the average total time of all plans that produce a particular number of results. For example, all DP plans for query Q4 that produce 140 results have an average total query time of 7.1s.

First, we note that while DP and DPU varies in planning time, their total time performances are comparable. That is, while DP needs more time for planning, this overhead is compensated by the faster execution that could be achieved through more optimal plans.

Most importantly, our systems DP, DPU and DPS, which optimize for both objectives, indeed enable different trade-offs between the two, i.e., reduce total processing time, when fewer results are needed. We can see for both Q1 and Q4, there is a trend that total times increase with the number of results. These trade-offs are not possible with the baseline systems RK and RD. Because the plans they produce are not Pareto-optimal but cost-optimized, the time performance of RK and RD is rather constant and does not change or correlate with the number of results. RD is particularly cost-optimized: while it achieves best performance, the plans it employs do not yield the desired number of results, e.g. none of its plans produces more than 7 results for Q1.

Further, there are obvious differences between cardinality and cost estimated for the plans and the actual number of results produced and time required by them. Many of the plans that are Pareto-optimal according to estimates, actually produce no results. These plans however, take the longest time to finish. This explains while the trend mentioned above is not clear, i.e., fewer results require less time but empty results require longest time.

Despites the simple estimates we employed in this work, the planning overhead can be outweighed by faster execution when the number of results is limited, i.e., DP and DPU provide better performance than the RK baseline. For example, to produce 7 results for Q1, DP and DPU require only 35% of the time RK needed to produce the same amount of result. Similarly, for Q4, DP and DPU requires only 28% of the total time of RK when 205 results have to be produced.

Summary. This experiment shows that compared to our work, the cost-based baselines produce only a small fraction of Pareto-optimal plans. The planning overhead incurred by our solution is relatively small compared to the gain in Pareto-optimality, e.g. 55 times more Pareto-optimal plans at the cost of 3 times higher planning cost for DPU. This Pareto-optimal planning has an effect on processing time and the actual results produced: using cost-optimized plans, the baselines cannot achieve the trade-off between cost and cardinality, while our solution reduces total processing time when fewer results are needed. This translates to about 4 times faster average performance than the RK baseline, when no more than 250 results are needed.

7 Conclusion

We propose the first solution towards a systematic optimization of Linked Data query processing, which considers both standard query operators and the specific characteristics of Linked Data source selection. The optimization result is the Pareto-set of optimal plans, representing different trade-offs between optimization objectives such as cost and cardinality. In experiments we compare our solution to cost-oriented baselines that independently optimize source selection and the processing of queries. Most plans computed by these baselines are sub-optimal such that the trade-off between different objectives is not adequately reflected. That is, while some baselines' plans achieve good time performance, they cannot produce the desired number of results; or they cannot help to improve time performance, given only a limited number of results are needed. Our solution provides different optimal tradeoffs, enabling several times reduction of processing cost when the number of results needed is less than 250.

As future work, we note from the experiments that a simple approximation of the proposed bounds can provide better planning performance, without compromising too much on the optimality of plans. We will study different approximations for the proposed DP solution.

References

[1] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, 2001.

[3] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K. Sattler, and J. Umbrich. Data summaries for on-demand queries over linked data. In *Proceedings of the 19th International Conference on World Wide Web*, Raleigh, North Carolina, USA, 2010.

[4] O. Hartig. Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, 2011.

[5] O. Hartig, C. Bizer, and J. Freytag. Executing SPARQL queries over the web of linked data. In *The Semantic Web - ISWC 2009*, pages 293–309. 2009.

[6] H. Huang and C. Liu. Selectivity estimation for SPARQL graph pattern. In *Proceedings of the 19th international conference on World wide web*, WWW '10, page 1115–1116, New York, NY, USA, 2010.

[7] D. Kossmann and K. Stocker. Iterative dynamic programming: a new class of query optimization algorithms. *ACM Trans. Database Syst.*, 25(1):43–82, 2000.

[8] G. Ladwig and T. Thanh. Linked data query processing strategies. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*, 2010.

[9] G. Ladwig and T. Tran. SIHJoin: Querying Remote and Local Linked Data. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, 2011.

[10] A. Levy, A. Rajaraman, J. Ordille, et al. Querying heterogeneous information sources using source descriptions. In *Proceedings of the International Conference on Very Large Data Bases*, 1996.

[11] T. Neumann. *Efficient Generation and Execution of DAG-Structured Query Graphs.* PhD thesis, 2005.

[12] T. Neumann and G. Weikum. Scalable join processing on very large RDF graphs. In *Proceedings of the 35th SIGMOD international conference on Management of data*, pages 627–640, Providence, USA, 2009. ACM.

[13] Z. Nie and S. Kambhampati. Joint optimization of cost and coverage of query plans in data integration. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, page 223–230, New York, NY, USA, 2001. ACM.

[14] C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, page 52–59, New York, NY, USA, 2001. ACM.

[15] R. Pottinger and A. Halevy. MiniCon: a scalable algorithm for answering queries using views. *The VLDB Journal*, 10:182–198, Sept. 2001.

[16] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. In *International Semantic Web Conference (1)*, pages 585–600, 2011.

[17] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34, Boston, Massachusetts, 1979. ACM.

[18] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: a wide-area distributed database system. *The VLDB Journal*, 5(1):048–063, Jan. 1996.

[19] B. Vance and D. Maier. Rapid bushy join-order optimization with cartesian products. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, page 35–46, New York, NY, USA, 1996. ACM.