Efficient Evaluation of Conjunctive Regular Path Queries Using Multi-way Joins

Nikolaos Karalis[0000−0002−0710−7180], Alexander Bigerl[0000−0002−9617−1466],
Liss Heidrich[0009−0006−2031−2548], Mohamed Ahmed Sherif[0000−0002−9927−2203],
and Axel-Cyrille Ngonga Ngomo[0000−0001−7112−3516]

DICE group, Department of Computer Science, Paderborn University, Germany
{nikolaos.karalis, alexander.bigerl, liss.heidrich, mohamed.sherif,
axel.ngonga}@uni-paderborn.de

Abstract. Recent analyses of real-world queries show that a prominent type of queries is that of conjunctive regular path queries. Despite the increasing popularity of this type of queries, only limited efforts have been invested in their efficient evaluation. Motivated by recent results on the efficiency of worst-case optimal multi-way join algorithms for the evaluation of conjunctive queries, we present a novel multi-way join algorithm for the efficient evaluation of conjunctive regular path queries. The hallmark of our algorithm is the evaluation of the regular path queries found in conjunctive regular path queries using multi-way joins. This enables the exploitation of regular path queries in the planning steps of the proposed algorithm, which is crucial for the algorithm's efficiency, as shown by the results of our detailed evaluation using the Wikidata-based benchmark WDBench. The results of this evaluation also show that our approach achieves a value of query mixes per hour that is 4.3 higher than the state of the art and that it outperforms all of the competing graph storage solutions in almost 70% of the benchmark's queries.

Keywords: knowledge graphs · conjunctive regular path queries · multi-way joins

1 Introduction

The ability to express queries requesting paths of arbitrary length between entities of a knowledge graph, also known as *(two-way) regular path queries* (RPQs) [3, 10], is a unique characteristic of graph query languages [18]. Finding paths of arbitrary length is crucial for many applications on knowledge graphs, such as path-based fact checking [27] and class expression learning [14]. As a result, many recent works have focused on developing approaches for the efficient evaluation of RPQs (e.g., [4, 6, 30]). However, most of these works do not take the fact that RPQs are usually part of more complex queries into account [1]. In fact, the results of two detailed studies of Wikidata's query logs [11, 21] show that *conjunctive two-way regular path queries* (C2RPQs), which are conjunctive queries extended with regular path queries [11], have received a lot of attention recently and are often used in practice.

Recently, Cucumides et al. [13] performed a theoretical analysis of the evaluation of C2RPQs using worst-case optimal multi-way join algorithms. Worst-case optimal multi-way join algorithms [22] are a recent advancement in query processing and have achieved state-of-the-art performance in evaluating conjunctive queries. Among other contributions, Cucumides et al. proposed the algorithm GenericJoinCRPQ that evaluates C2RPQs using multi-way joins. However, as previous theoretical analyses of such algorithms, they do not discuss the involvement of RPQs in aspects of multi-way joins that are crucial for their efficiency in practice. First, they assume a given variable ordering. Second, given a particular variable, they do not discuss the order in which operations should be carried out. For example, provided the SPARQL graph pattern {`?x <p1> <o1> . ?x <p2> <o2> . ?x <p3>* <o3> .`}, should the RPQ be evaluated before or after the join operation between the first two triple patterns? To the best of our knowledge, there have not been any implementations of multi-way join algorithms for the evaluation of C2RPQs.

In this work, we hence focus on presenting a novel multi-way join algorithm for the evaluation of C2RPQs over RDF graphs using SPARQL. The main characteristic of our approach is that it evaluates RPQs using multi-way joins. This evaluation (1) enables the integration of RPQs in multi-way join plans; our approach is generic and can be integrated in any system supporting multi-way joins, (2) allows for the evaluation of C2RPQs without the need for materializing the results of RPQs—which can be large—and (3) enables the accurate size estimation of RPQs, as it allows for their evaluation up to an arbitrary depth, and thus allows their inclusion in optimization steps of the evaluation, such as the variable ordering process. The proposed algorithm is implemented in a state-of-the-art triple store supporting worst-case optimal multi-way joins. We evaluate our approach using WDBench [2]. We compare the performance of our approach against the performance of multiple state-of-the-art graph storage solutions. The results of this comparison suggest that our solution is on average 4.3 times faster than the second best system. We also carry out a detailed evaluation of different execution strategies. Its results show the importance of including RPQs in the optimization steps of the proposed algorithm. Like a number of approaches for the evaluation of conjunctive queries based on worst-case optimal joins [5, 8, 19], the performance of our approach depends on the order of variables and the order in which operations are carried out for a particular variable.

The rest of the paper is structured as follows. In Section 2, we provide background knowledge on the topics that are covered in this paper. We discuss related works in Section 3. We present our approach for the evaluation of C2RPQs using multi-way joins in Section 4. Our experimental results are presented in Section 5. In Section 6, we conclude and discuss possible future research directions.

2 Preliminaries

Below, we cover the features of SPARQL that are relevant to this work and briefly summarize worst-case optimal multi-way joins and GenericJoinCRPQ [13].

2.1 RDF and SPARQL

The semantics of SPARQL have been extensively covered in previous works (e.g., [20, 24–26]). Here, following standard notation, we recapitulate the semantics and properties of the features of the language that are used later in this work. More specifically, we cover *basic graph patterns*, *union graph patterns* and *property path patterns*. The formal definitions presented below rely on the following pairwise disjoint sets. Let **I** be an infinite set of IRIs, **B** an infinite set of blank nodes, and **L** an infinite set of literals. Furthermore, let **V** be an infinite set of variables. An RDF graph is a set of *subject-predicate-object* triples and is formally defined as $G = \{(s, p, o) \mid s \in (\mathbf{I} \cup \mathbf{B}), p \in \mathbf{I}, o \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})\}$.

Basic and Union Graph Patterns [19, 24] A basic graph pattern (BGP) is a set of triples and is formally defined as $P = \{(s, p, o) \mid s \in (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V}), p \in (\mathbf{I} \cup \mathbf{V}), o \in (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})\}$. An element of P is called a triple pattern and is denoted as *tp*. As in [19, 24], we do not consider blank nodes in triple patterns because they behave as variables. The set of variables of a triple pattern is denoted as *var(tp)*. BGPs are conjunctive queries and their semantics are defined using mappings. A mapping is a partial function assigning RDF terms (i.e., IRIs, blank nodes, or literals) to variables and is formally defined as $\mu : \mathbf{V} \to (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$. The set of variables, over which a mapping μ is defined, is called the domain of μ and is denoted as $dom(\mu)$. Provided a triple pattern *tp*, $\mu(tp)$ denotes the RDF triple obtained by replacing every variable in *var(tp)* with its corresponding value in μ. Two mappings μ_1 and μ_2 are compatible, if and only if for every variable $?v \in dom(\mu_1) \cap dom(\mu_2)$ holds that $\mu_1(?v) = \mu_2(?v)$. The join and union operations between two sets of mappings Ω_1 and Ω_2 are defined as:

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible}\} \text{ and}$$
$$\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}.$$

A triple pattern *tp* and a BGP P are evaluated over an RDF graph G as follows:

$$[\![tp]\!]_G = \{\mu \mid dom(\mu) = var(tp) \text{ and } \mu(tp) \in G\} \text{ and}$$
$$[\![P]\!]_G = [\![tp_1, \ldots, tp_n]\!]_G = [\![tp_1]\!]_G \bowtie \ldots \bowtie [\![tp_n]\!]_G.$$

In SPARQL, braces allow us to form different graph patterns. Both BGPs and triple patterns are graph patterns. The concatenation (conjunction) and union of two graph patterns P_1 and P_2 are denoted as $(P_1\ AND\ P_2)$ and $(P_1\ UNION\ P_2)$, respectively, and they are evaluated as follows:

$$[\![P_1\ AND\ P_2]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \text{ and } [\![P_1\ UNION\ P_2]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G.$$

A graph pattern P is in *UNION* normal form if it is in the form $(P_1\ UNION\ \ldots\ UNION\ P_n)$ and each P_i, for $1 \leq i \leq n$, is *UNION*-free. A graph pattern is *UNION*-free, if it does not contain any union graph patterns.

Theorem 1 (Existence of UNION normal form [24]). *Every graph pattern P using the AND and UNION operators is equivalent to a graph pattern P', which is in UNION normal form.*

Property Paths Patterns [20] In SPARQL, two-way RPQs are expressed as property path patterns. A property path pattern is a triple $r = (s, e, o)$, where $s, o \in (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$ and e is constructed by the following grammar:

$$e := \alpha \mid e^{-} \mid e \cdot e \mid e + e \mid e^{+} \mid e^{*} \mid e? \ , \ \alpha \in \mathbf{I}.$$

The set of variables of a property path pattern r is denoted as *var(r)*. For simplicity, we assume that property paths do not have the same variable in both the subject and object position. The implementation of our approach (Section 4) supports RPQs having the same variable in the subject and object position. Note that we omit the rules of negated property sets from the grammar. As in [1], we do not consider negated property sets in this work. Expressions of the shape $e_1 \cdot e_2$ and $e_1 + e_2$ can be rewritten as joins and unions, respectively (see Example 1) [17, 25]. Consequently, property path patterns using only the first four rules of the grammar can be rewritten to equivalent graph patterns consisting only of basic and union graph patterns and be evaluated as shown above. Property paths using the transitive closure operators $*$ or $+$ are evaluated under set semantics. The expression $e?$ is a special case of e^*; it returns solutions for paths of length 0 and length 1. From this point on, we will use the term RPQ only for property paths using any of the $+$, $*$, or ? operators, as the remaining expressions can be rewritten as graph patterns without property paths.

Example 1. The queries provided below are semantically equivalent.

```
Q1: SELECT ?s WHERE {?s (<e1>/(<e2>/<e3>)+)|(<e4>/<e5>) <o>}
Q2: SELECT ?s WHERE {{?s <e1> ?t . ?t (<e2>/<e3>)+ <o>} UNION
                    {?s <e4> ?v . ?v <e5> <o>}}
```

Conjunctive Two-way Regular Path Queries In SPARQL, a C2RPQ is a pattern that only uses triple patterns, the operator *AND*, and RPQs [11]. In $Q2$ of Example 1, the first graph pattern of the *UNION* is a C2RPQ consisting of a triple pattern and an RPQ, whereas the second graph pattern is a BGP. As the property path patterns that we consider in this work use only the *AND* and *UNION* operators, graph patterns comprised of basic, union and property path patterns can be rewritten to a graph pattern in *UNION* normal form [25]. Our approach presented below deals with graph patterns that are in *UNION* normal form, where each union operand is either a C2RPQ or a BGP (e.g., $Q2$). Our implementation applies the *UNION* normal form to queries, while parsing them.

2.2 Worst-case Optimal Multi-way Joins

Worst-case optimal multi-way join algorithms [22] satisfy the AGM bound [7], i.e., their runtime complexity is bounded by the worst-case size of the result of the input query [19]. Since their recent introduction, they have gained a lot of attention and, in particular, have achieved state-of-the-art performance when evaluating graph pattern queries (e.g., [5, 8, 15, 19]). Unlike pair-wise joins that carry out join operations on two join operands at a time, worst-case optimal

Algorithm 1 Generic Join for Basic Graph Patterns

```
1: // Generator function: execution is resumed after a yield operation
2: function GenericJoin($P$, $G$, $X$)        ▷ $P$: BGP, $G$: RDF Graph, $X$: Mapping
3:     if all variables are resolved then yield $X$ and return
4:     $?x \leftarrow$ select an unresolved variable from $X$
5:     $K \leftarrow \bigcap_{tp \in P \mid ?x \in var(tp)} \{\mu(?x) \mid \mu \in [\![tp]\!]_G\}$
6:     for all $k \in K$ do
7:         $X(?x) \leftarrow k$ ; $P' \leftarrow$ assign $k$ to all occurrences of $?x$ in $P$
8:         yield all GenericJoin($P'$, $G$, $X$) ▷ after yielding proceeds with the next $k$
```

multi-way join algorithms evaluate queries recursively on a per variable basis [15, 19]. This evaluation method does not store any intermediate results and allows for mappings to be written to the result incrementally. A worst-case optimal multi-way join algorithm based on Generic Join [23] is shown in Algorithm 1. In practice, the performance of Generic Join is mostly affected by the order of variables (line 4) [19] and the set intersection, which finds the possible values of the selected variable (line 5). In fact, the set intersection should be guided by the triple pattern with the smallest set of values for the selected variable [29]. Last, indices (e.g., [5, 8]) also play an important role in the efficiency of worst-case optimal multi-way join algorithms [15].

2.3 GenericJoinCRPQ

As mentioned, Cucumides et al. [13] perform a theoretical analysis of the evaluation of C2RPQs using worst-case optimal algorithms. In their work, the authors obtain size bounds for several classes of C2RPQs and show that there are C2RPQs that cannot be evaluated by a worst-case optimal algorithm. Despite their latter finding, the authors devise algorithms based on existing worst-case optimal algorithms. One of these algorithms is GenericJoinCRPQ, which is an extension of Generic Join (Algorithm 1). More specifically, GenericJoinCRPQ also considers the RPQs of the graph pattern that have the selected variable $?x$ for the set intersection (line 5). Additionally, GenericJoinCRPQ materializes the RPQs that have the selected variable $?x$, once $?x$ is replaced with a value k (line 7). In subsequent steps, materialized RPQs are treated as triple patterns. However, the materialization of RPQs can be avoided by evaluating them with multi-way joins. As the authors study the complexity of GenericJoinCRPQ theoretically, they assume a given variable ordering (line 4). As discussed, the variable ordering is crucial for the performance of multi-way join algorithms. Hence, it is necessary to consider RPQs in the variable selection process. Furthermore, the authors assume an arbitrary order for the set intersection (line 5). In practice, should a triple pattern or an RPQ guide the set intersection? Last, the authors argue that the running time of GenericJoinCRPQ might end up being too high when there are multiple recursive steps evaluating RPQs. Our experimental results show that, even in such cases, the evaluation of C2RPQs with multi-way joins outperforms existing solutions by considering RPQs in planning steps.

3 Related Work

Multiple algorithms have been proposed for the evaluation of RPQs. A type of approaches that has received a lot of attention is that based on finite automata [6, 28, 30] and recently, an approach based on matrix algebra was introduced [4]. As existing SPARQL engines (e.g., Blazegraph and Virtuoso), our work falls into the category of approaches that use existing relational operations for the evaluation of RPQs [1]. However, to the best of our knowledge, our work is the first to use multi-way joins for the evaluation of RPQs. In addition to the specialized evaluation algorithms, specialized indices have also been proposed for the efficient evaluation of RPQs (e.g., [6, 16]). We refer the reader to [6] for a more detailed review of the literature on the evaluation of RPQs.

For the efficient evaluation of C2RPQs, the works in the literature mostly focus on obtaining accurate cardinality estimations of RPQs that ultimately lead to good orderings of two-way joins [1]. A recent work in this direction is presented in [1]. In [1], the authors propose a cost model for RPQs and an approach based on random walks for estimating the size of RPQs that do not have any nested transitive closures (i.e., for path expressions α^+ and α^*). Given an RPQ, a fixed number of random walks evaluate the RPQ up to a specified depth and the RPQ's estimated cost is ultimately equal to the sum of the number of results returned by each random walk. A shortcoming of this approach is that random walks use bag semantics instead of set semantics, which, as per the authors, might lead to overestimated cardinalities in dense graphs. Our approach presented below proposes an end-to-end evaluation methodology for C2RPQs based on multi-way joins and follows set semantics for estimating the size of RPQs.

4 Evaluating C2RPQs with Multi-way Joins

In this section, we present our approach for the evaluation of C2RPQs using multi-way joins. Note that the proposed algorithm is not worst-case optimal [13]. The main characteristic of our approach is the evaluation of RPQs found in C2RPQs using multi-way joins. The evaluation of RPQs using multi-way joins offers multiple benefits. First, it allows for the easy integration of RPQs into multi-way join plans. Our approach is generic and can be adopted by any system supporting multi-way joins. Second, it does not require the materialization of RPQs. As in multi-way joins for conjunctive queries, we generate the results of RPQs incrementally. However, as we discuss later, there are cases where the materialization of RPQs leads to an improved performance. Third, it allows for the accurate estimation of the size of RPQs up to an arbitrary depth. The accurate size estimation of RPQs, enables their inclusion in planning steps of the algorithm, which is crucial for the efficient evaluation of C2RPQs [1].

4.1 Evaluation of RPQs

The SPARQL standard [17] defines the function ALP for the evaluation of transitive closures (i.e., path patterns using the * or + operators). Consider an RPQ

$r = (s, \alpha^+, ?o)$, with $s \notin \mathbf{V}$. At depth 1, ALP evaluates the triple pattern $tp_1 = (s, \alpha, ?o)$. At depth 2, ALP evaluates the triple pattern $tp_2 = (u, \alpha, ?o)$ for every term $u \in \{\mu(?o) \mid \mu \in [\![tp_1]\!]_G\}$. In general, for $i > 1$, ALP uses the terms generated at depth $(i - 1)$—that have not been visited already—to evaluate a triple pattern tp_i at depth i. This is generalized to BGPs and unions of BGPs by replacing α^+ with e^+ in r, provided e does not have any nested transitive closures. Here, the main observation is that a transitive closure is evaluated by a recursive procedure, which, in turn, evaluates a graph pattern P that does not have any RPQs at each recursive step [1]. We leverage this observation and use multi-way joins to evaluate P. If P is a BGP, we simply use Generic Join. If P is a union graph pattern, it can be rewritten to an equivalent pattern P' in *UNION* normal form (Section 2.1, Theorem 1). The BGPs of P' are then evaluated independently and in a serialized manner. The UNION normal form might lead to a large number of joins; however, this is not common in practice. For nested transitive closures, ALP needs to evaluate a C2RPQ or a union of C2RPQs at each step. In such cases, we use Algorithm 3 (Section 4.2).

Algorithm 2 presents the evaluation of RPQs using multi-way joins. The starting point of the evaluation is EvalRPQ. Based on the number of variables that the provided RPQ has, EvalRPQ calls the appropriate function for its evaluation. EvalRPQ_TV (TV stands for term and variable) is based on the ALP function and is called for RPQs that have only one variable (line 5). Here, we assume that the object of the RPQ is the variable. The function works in the same manner for the case of the subject being the variable. As their names suggest, the list `to_visit` and the set `visited` keep track of the nodes of the input RDF graph that need to be visited and have already been visited, respectively. EvalRPQ_TV checks first if the RPQ should return paths of length 0 (lines 14-17). This is the case for the $*$ and ? operators. As discussed above, an RPQ that does not have any nested RPQs can be rewritten to a graph pattern that does not have any property path patterns after removing its transitive closure operator. The resulting graph pattern P' (line 18) is used to evaluate paths of length greater than zero (lines 20-29). The while-loop runs until there are no more nodes left to be visited. The term in P' corresponding to the original term of the RPQ's subject position is replaced every time with the node that needs to be visited (line 23). The updated P' is then evaluated by Generic Join (line 24). Nodes that have already been visited are discarded (line 26). The RPQ's mapping is updated for each distinct node and returned as a result (line 29). If the `max_depth` is not exceeded, the nodes returned by Generic Join are pushed into `to_visit`. For the ? operator, `max_depth` is set to one. For the + and $*$ operators, it is set to the largest possible integer value.

We omit the pseudocode for EvalRPQ_VV (VV stands for variable and variable) and EvalRPQ_TT (TT stands for term and term) due to space considerations. As in the SPARQL standard [17], EvalRPQ_VV assigns every node of the input graph to the subject position of the RPQ and calls EvalRPQ_TV. In fact, for the + operator, it restricts the visited nodes using the first IRI of the property path expression. EvalRPQ_TT treats either the subject or the object

Algorithm 2 Evaluation of RPQs with Multi-way Joins

```
 1: // The pseudocode for EvalRPQ_TT and EvalRPQ_VV is omitted for brevity
 2: // As in the SPARQL standard [17], it relies on EvalRPQ_TV
 3: function EvalRPQ($r$, $G$, $X$)      ▷ $r$: RPQ, $G$: RDF Graph $X$: Mapping
 4:     if $|var(r)| = 1$ then
 5:         yield all EvalRPQ_TV($r$, $G$, $X$)
 6:     else if $|var(r)| = 0$ then
 7:         if EvalRPQ_TT($r$, $G$, $X$) is true then yield $X$
 8:     else if $|var(r)| = 2$ then
 9:         yield all EvalRPQ_VV($r$, $G$, $X$)
10: function EvalRPQ_TV($r$, $G$, $X$)
11:     // For brevity, we only cover the case of the subject being known
12:     to_visit ← [(subject, 0)]                     ▷ List of (term, depth) pairs
13:     visited ← {}                                  ▷ Set of terms
14:     if paths of length 0 need to be returned then ▷ * or ? operator
15:         $X$(object_var) ← subject                 ▷ mapping is updated
16:         yield $X$                                 ▷ The mapping is yielded
17:         insert subject into visited               ▷ keep track of visited terms
18:     $P'$ ← graph pattern corresponding to $r$ without the transitive closure
19:     $X'$ ← empty solution mapping for $P$
20:     while to_visit is not empty do
21:         (term, depth) ← last item from to_visit
22:         remove the last item from to_visit
23:         replace the value of subject with the value of term in $P'$
24:         for all GenericJoin($P'$, $G$, $X'$) do
25:             object ← $X'$(object_var)
26:             if visited contains object then continue
27:             $X$(object_var) ← object ; insert object into visited
28:             yield $X$                 ▷ Output the updated mapping of the RPQ $r$
29:             if depth+1 < max_depth then push (object, depth+1) into to_visit
```

as a variable and then calls EvalRPQ_TV. If EvalRPQ_TV yields the term that was replaced by the variable, it returns true; otherwise it returns false. To avoid clutter, we assume that P' is in *UNION* normal form and that the BGPs comprising P' are evaluated one after another by Generic Join (line 26). To deal with nested RPQs, we use Algorithm 3 (Section 4.2) instead of Generic Join.

4.2 Evaluation of C2RPQs

Algorithm 3 presents our approach for the evaluation of C2RPQs. One of the main characteristics of our algorithm is that it does not consider in any part of the evaluation RPQs that have two variables. Such RPQs are considered once their subject or object position is bounded to a particular value in one of the recursive steps of the algorithm. The motivation behind this choice is twofold. First, RPQs having only one variable are evaluated more efficiently; EvalRPQ does not have to iterate over unnecessary nodes of the provided graph. Second,

Algorithm 3 Evaluation of C2RPQs with Multi-way Joins

```
 1: function EvalC2RPQ($Q$, $G$, $X$)     ▷ $Q$: C2RPQ, $G$: RDF Graph $X$: Mapping
 2:     if there are no RPQs in $Q$ then
 3:         yield all GenericJoin($Q$, $G$, $X$) and return
 4:     for all RPQs $r \in Q, |var(r)| = 0$ do     ▷ Evaluate RPQs having no variables
 5:         if EvalRPQ_TT($r$, $Q$, $X$) is false then return
 6:         else remove $r$ from $Q$
 7:     if $Q$ is empty then yield $X$ and return     ▷ A solution is found
 8:     $?x \leftarrow$ select an unresolved variable from $X$     ▷ Uses RPQs and triple patterns
 9:     $q \leftarrow$ PrioRPQ($Q$, $?x$)     ▷ Check if an RPQ should guide the set intersection
10:     if $q$ is a triple pattern then     ▷ Set intersection only between triple patterns
11:         $K \leftarrow \bigcap_{tp \in Q | ?x \in var(tp)} \{\mu(?x) \mid \mu \in [\![tp]\!]_G\}$     ▷ Set intersection guided by $q$
12:         for all $k \in K$ do
13:             $X(?x) \leftarrow k$ ; $Q' \leftarrow$ assign $k$ to all occurrences of $?x$ in $Q$
14:             yield all EvalC2RPQ($Q'$, $G$, $X$) and return
15:     for all EvalRPQ($q$, $G$, $X$) do     ▷ $q$ is an RPQ, set intersection guided by $q$
16:         if $X(?x) \notin \{\mu(?x) \mid \mu \in [\![tp]\!]_G\}$ for any $tp \in Q, ?x \in var(tp)$ then continue
17:         $Q' \leftarrow$ assign $k$ to all occurrences of $?x$ in $Q$ and remove $q$ from $Q$
18:         yield all EvalC2RPQ($Q'$, $G$, $X$)
19: function PrioRPQ($Q$, $?x$)     ▷ See Section 4.3, Set Intersection
20:     // Checks if the set intersection should be guided by an RPQ or a triple pattern
21:     // Considers all RPQs $r$, for which $?x \in var(r)$ and $|var(r)| = 1$
```

we are able to acquire more accurate size estimates for the remaining variable. RPQs having two variables are evaluated only if they do not participate in any join operations with triple patterns or are part of a cross product. We do not cover such cases, as they cannot be optimized.

The core of our algorithm is the function EvalC2RPQ. If the input query is not a C2RPQ, EvalC2RPQ simply calls Generic Join (lines 2-3). EvalC2RPQ prioritizes RPQs that do not have any variables, as they are not subject to any further changes. (line 4-6). If the evaluation of all RPQs having no variables returns true, EvalC2RPQ proceeds with the evaluation of the remaining C2RPQ. As in Generic Join, EvalC2RPQ first selects the variable to be resolved (line 8). Here, C2RPQ considers triple patterns and RPQs having only one variable. The variable selection strategies are detailed in Section 4.3. Once a variable is selected, EvalC2RPQ uses PrioRPQ to find out whether an RPQ or a triple pattern should guide the set intersection (line 9). Again, when it comes to RPQs, PrioRPQ considers only RPQs having one variable. If a triple pattern is selected, EvalRPQ behaves as Generic Join. If an RPQ is selected, it is evaluated using EvalRPQ (lines 15-18). For every value $X(?x)$ returned by EvalRPQ, EvalC2RPQ checks if $X(?x)$ is found in the evaluation of all triple patterns *tp*, with $?x \in$ *var(tp)* (line 16). Values that are not found in any of the evaluations, are discarded. RPQs having only the selected variable that are not evaluated in this recursive step will be evaluated in the subsequent step (lines

4-6). For example, if there are two RPQs that can be evaluated for the selected variable, at least one of them will be evaluated in the following recursive step.

4.3 Query Planning and Optimizations

Size Estimation of RPQs Estimating the size of RPQs (i.e., the number of solutions they return) enables their consideration in the planning steps of the algorithm. Recall that in our algorithm, in planning steps, we consider only RPQs that have one variable. To estimate the size of such RPQs, we evaluate them up to a specified depth. This is possible by assigning a particular value to `max_depth` in EvalRPQ_TV (Algorithm 2). Note that the returned estimation is equal to the number of solutions found until the provided depth. Following [1], we evaluate RPQs up to depth 5 to estimate their size. Henceforth, we will refer to this estimation as *default estimation*. For RPQs having property path expressions that consist of a single term (e.g., $(s, a^+, ?o)$, with $s \notin \mathbf{V}$), we introduce the *shallow estimation*. In such cases, the shallow estimation is equal to the evaluation's size of the triple pattern that results after removing the transitive closure operator (i.e., $(s, a, ?o)$, with $s \notin \mathbf{V}$). The size of such triple patterns are provided in constant time by indices used for multi-way joins [5, 8]. As discussed below, if possible, we use the shallow estimation to avoid using the default estimation, which is more computationally expensive. Last, to avoid computing the estimation of an RPQ for a particular subject or object multiple times, we cache the estimated size for each subject and object.

Set Intersection In Generic Join, the triple pattern that has the smallest cardinality for the selected variable should guide the search for finding the variable's possible values [29]. For C2RPQs, we also need to consider the RPQs that have the selected variable. In EvalC2RPQ (line 9, Algorithm 3), we first find the minimum cardinality of the selected variable among the triple patterns (see Variable Ordering). Then, for each RPQ, we first calculate its shallow estimation, if possible. If the shallow estimation is greater than the minimum cardinality, we do not calculate the default estimation, as the RPQ will not guide the set intersection. If the shallow estimation is lower than the minimum cardinality, we calculate the default estimation to get a more accurate estimation and update the minimum cardinality accordingly. In the end, the RPQ or the triple pattern having the minimum cardinality for the selected variable guides the set intersection.

Variable Ordering The order in which variables are resolved is imperative for the efficiency of multi-way joins [19]. The order can be static [5, 19] or dynamic [8]. In the first case, the variable ordering of a query is determined before the query's evaluation. In the second case, the variable to be resolved is selected at each recursive step. As described above, the proposed algorithm considers only RPQs with one variable at planning steps. With a static variable ordering, the algorithm would have to completely disregard RPQs that have two variables at the beginning of a query's evaluation. For this reason, our algorithm uses a dynamic variable ordering. During the evaluation of C2RPQs, as variables are recursively resolved, RPQs that start with two variables end up at some point having only one variable and hence, can be considered for the variable ordering.

For selecting a variable at each recursive step, we experiment with two strategies. The first strategy was proposed in [8] and at each recursive step, it selects the variable that has the largest guaranteed *reduction* of the search space spanned by Cartesian products of the triple patterns' solutions. We refer to this strategy as *reduction factor*. The second strategy is based on the one proposed in [5] and at each step, it selects the variable that has the *minimum cardinality* (i.e., the variable that is estimated to have the smallest set of possible values). For both strategies, the size of RPQs is estimated as described in the set intersection.

Materialization of RPQs By using multi-way joins, our approach is able to evaluate C2RPQs without having to materialize the results of RPQs. However, there are cases where the materialization of RPQs improves the performance of the proposed algorithm. Consider the SPARQL query `SELECT * WHERE {` `?x <p1> ?z . ?z <p2> ?y . ?y <p3>+ <o>` `}`. If ?y is not the first variable to be evaluated by EvalC2RPQ, the RPQ might end up being evaluated multiple times in intermediate recursive steps (line 15, Algorithm 3), while always yielding the same results. To avoid unnecessary computations, we materialize the results of such RPQs, i.e., of RPQs that have one variable *before the start of the query evaluation* and are evaluated in intermediate recursive steps of EvalC2RPQ. Note that in [13], all RPQs are materialized.

5 Experimental Results

In this section, we present the performance evaluation of the proposed algorithm, which we have implemented in the tensor-based triple store Tentris [8, 9]. We refer to our implementation as TentrisRPQ. For the evaluation of TentrisRPQ, we used the the recently introduced benchmark WDBench [2]. WDBench consists of a real-world dataset based on Wikidata and queries that are extracted from Wikidata's query logs. The experiments presented below were carried out on a Debian 10 server with an AMD EPYC 7282 CPU, 256GB RAM and a 2TB Samsung 970 EVO Plus SSD. Supplementary material—including datasets, binaries, queries, scripts, and configurations—is available online.[1]

5.1 Systems and Experimental Setup

We compared the performance of TentrisRPQ against the performance of the following well-established triple stores: (i) Blazegraph 2.1.6.RC, (ii) Fuseki 4.9.0, (iii) GraphDB 10.3.3 (free version), and (iv) Virtuoso 7.2.10. In our experiments, we also included the graph database MilleniumDB[2] [28]. In MilleniumDB, BGPs

[1] https://github.com/dice-group/c2rpqs-benchmark

[2] https://github.com/MillenniumDB/MillenniumDB, commit: 442e650

are evaluated by a worst-case optimal multi-way join algorithm and RPQs are evaluated following an automaton-based approach. When it comes to C2RPQs, in MilleniumDB, “paths are pushed to the end of join plans” [28] and hence are not considered in multi-way join plans. We also carried out an evaluation of different execution strategies using multiple versions of TentrisRPQ.

Our experiments were carried over HTTP using the benchmark execution framework IGUANA 3.3.3 [12]. For the set of queries that we used in this work, we created a stress test that was executed by every system four consecutive times, with the first execution serving as a warm-up run. The query timeout was set to three minutes. The timeout was set in the systems’ respective configurations. As in previous works [9], we measured the performance of the systems using the following metrics: (i) QPS, i.e., the number of queries executed per second, (ii) pAvgQPS, i.e., the penalized average QPS and (iii) QMPH, i.e., the number of query mixes executed per hour. Queries that failed (e.g., timed out or returned an error code) are penalized with a runtime of three minutes.

5.2 Datasets and Queries

The dataset of WDBench is an extract of Wikidata containing 1.26B triples (92.4M distinct subjects, 8.6K distinct predicates, and 305M distinct objects). In this work, we focus on the set of C2RPQs provided by WDBench[3]. As in [1], we did not consider queries with cross products and queries not using any of the +, *, or ? operators. To ensure the fair comparison of the benchmarked systems, from the remaining queries, we only kept those queries, for which the systems that were able to evaluate them before the specified timeout returned the same number of solutions and bindings. For example, we had to discard the queries returning more than 2^{20} results, which is Virtuoso’s hard limit [2, 6, 8]. To alleviate this issue, some works in the literature (e.g., [2, 28]) restrict the number of solutions using SPARQL’s `LIMIT`. However, the use of `LIMIT` without ordering (`ORDER BY` in SPARQL) does not guarantee that the benchmarked systems return the same results. In addition, systems supporting multi-way joins—including ours—have an inherent advantage when `LIMIT` is used without ordering, as they do not have to compute the full result set; they can terminate once they have output the requested number of solutions. For these reasons, instead of using `LIMIT`, we opted for discarding queries as described above. Ultimately, we used 305 queries in our experiments, which, as in previous works (e.g., [1, 28]), were evaluated under set semantics (i.e., SPARQL’s `DISTINCT` is used). Note that no queries were discarded because of TentrisRPQ, as it returns the same amount of solutions with at least one other triple store in all queries.

5.3 Evaluation of Execution Strategies

To get insights on the impact that our proposed query planning and optimizations have on the evaluation of C2RPQs, we compared different execution strategies using different versions of TentrisRPQ, which are presented below.

[3] https://github.com/MillenniumDB/WDBench/blob/master/Queries/c2rpqs.txt

Table 1. The comparison of the different execution strategies. The column failed reports the number of queries for which the corresponding system failed (e.g., timed out) at least once.

	Warm Runs			Cold Run		
	QMPH	pAvgQPS	failed	QMPH	pAvgQPS	failed
TentrisRPQ^{FMC}	10.167	571.654	0	6.433	241.101	0
TentrisRPQ^{FRF}	1.304	587.925	5	1.195	222.749	5
TentrisRPQ^{RPC}	3.127	576.814	3	2.701	212.881	3
TentrisRPQ^{RP}	1.323	586.958	7	1.257	217.213	6
TentrisRPQ^{JP}	0.311	78.298	36	0.301	32.791	38

TentrisRPQ^{JP} (Join Prioritization) This version always prioritizes join operations, which leads to RPQs being evaluated only after their variables have been resolved. RPQs are not considered in the variable selection process.

TentrisRPQ^{RP} (RPQ Prioritization) This version always prioritizes RPQs with one variable. At each recursive step, an RPQ (alongside its remaining variable) is chosen to be evaluated using the shallow estimation. Note that the set intersection of a variable is always guided by an RPQ. If the shallow estimation is not applicable to any of the RPQs, we select the first available RPQ.

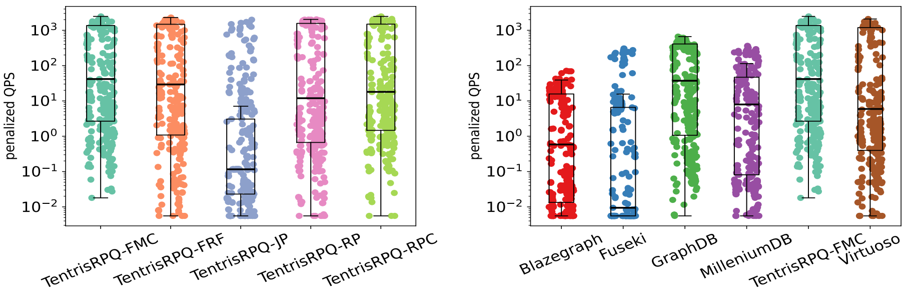
TentrisRPQ^{RPC} (RPQ Prioritization and Materialization) This version follows the same evaluation process as the one presented above and, in addition, materializes RPQs as discussed in Section 4.3.

TentrisRPQ^{FRF} (Full Version with Reduction Factor) This version fully implements the approaches presented in Section 4. For selecting variables, it follows the strategy based on the reduction factor. In fact, all of the previously presented versions of TentrisRPQ use the reduction factor strategy (Section 4.3) for selecting variables, which is the strategy supported by Tentris.

TentrisRPQ^{FMC} (Full Version with Minimum Cardinality) This version also fully implements the proposed approaches, but for selecting variables, it uses the minimum cardinality strategy (Section 4.3). The minimum cardinality strategy is not supported by Tentris; it is only part of TentrisRPQ.

The performance of each execution strategy is presented in Table 1 and Figure 1a. TentrisRPQ^{JP} achieves the lowest QMPH and pAvgQPS. The remaining versions of TentrisRPQ achieve similar pAvgQPS values, with TentrisRPQ^{FMC} achieving the best median penalized QPS followed by TentrisRPQ^{FRF}. Regarding the QMPH, TentrisRPQ^{FMC} achieves the best performance, which is 3.2 higher than the second best reported performance (TentrisRPQ^{RPC}).

These results show the importance of considering RPQs in the planning steps of the proposed algorithm. TentrisRPQ^{FMC}—which is the full version of TentrisRPQ and uses the minimum cardinality estimation—does not time out in any of



(a) Comparison of execution strategies (b) Comparison with existing solutions

Fig. 1. Performance of the benchmarked systems, including the different execution strategies of TentrisRPQ, in terms of penalized QPS (warm runs).

the queries and achieves the highest QMPH value, outperforming all of the three versions of TentrisRPQ that prioritize a particular execution strategy. This is not the case for the second full version of TentrisRPQ, namely TentrisRPQ^{FRF}, that uses the reduction factor estimation for selecting variables. The difference between the QMPH values reported by TentrisRPQ^{FMC} and TentrisRPQ^{FRF} highlights once again the importance of selecting a good variable ordering. The reduction factor strategy leads TentrisRPQ^{FRF} to time out in some queries, which has a negative impact on its QMPH value. However, after closely examining the results, we found queries for which TentrisRPQ^{FRF} reports a more than ten times higher QPS. In these queries (e.g., the queries 53 and 186 in our supplementary material), the minimum cardinality prioritizes variables that do not participate in join operations and have values with large multiplicities. As a result, the set of possible values of the join variables that are found in triple patterns with the prioritized variables are not restricted enough. This leads to the conclusion that, as part of our future work, we need to devise a variable selection strategy that combines both of the strategies used in this work. Last, the results of TentrisRPQ^{RP} and TentrisRPQ^{RPC} show that the proposed materialization of RPQs improves the performance of our algorithm.

5.4 Comparison with Existing Solutions

To compare TentrisRPQ with existing storage solutions, we use TentrisRPQ^{FMC}. The comparison's results are shown in Table 2 and Figure 1b. TentrisRPQ^{FMC} achieves the highest QMPH value, which is 4.3 times higher than the second best value (GraphDB). TentrisRPQ^{FMC} also achieves the highest pAvgQPS and median QPS. Virtuoso and GraphDB achieve the second best values for pAvgQPS and median QPS, respectively.

TentrisRPQ^{FMC} outperforms all of the competing systems in 213 out of 305 queries. More specifically, it achieves at least two and five times higher penalized QPS than every other system in 81 and 27 queries, respectively. To

Table 2. The comparison of different systems. The column failed reports the number of queries for which the corresponding system failed (e.g., timed out) at least once.

	Warm Runs			Cold Run		
	QMPH	pAvgQPS	failed	QMPH	pAvgQPS	failed
Blazegraph	0.257	8.315	49	0.256	7.733	48
Fuseki	0.129	19.672	145	0.128	12.932	145
GraphDB	2.322	184.851	1	2.110	80.443	1
MilleniumDB	0.754	48.034	9	0.728	39.395	10
TentrisRPQ^{FMC}	10.167	571.654	0	6.433	241.101	0
Virtuoso	1.104	450.330	10	1.098	124.812	10

find the shortcomings of our approach, we focused on the queries for which TentrisRPQ^{FMC} achieves at least 2 times lower penalized QPS than any other system; this is the case for 45 queries. In these queries, TentrisRPQ^{FMC} is mostly outperformed by the competing systems due to a bad variable ordering or due to the computational overhead introduced by the default size estimation of RPQs. This is justified by the fact that, in the majority of these queries, another version of TentrisRPQ achieves the overall best performance. For example, in most of the queries where MilleniumDB outperforms TentrisRPQ^{FMC}, TentrisRPQ^{JP} performs better than or similar to MilleniumDB. Note that TentrisRPQ^{JP} and MilleniumDB follow similar execution strategies.

6 Conclusion and Future Work

We presented an approach for the efficient evaluation of C2RPQs using multi-way joins. By evaluating RPQs found in C2RPQs using multi-way joins, the proposed algorithm is able to exploit RPQs in the planning steps of multi-way joins, which, as demonstrated by our detailed evaluation of different execution strategies, is crucial for its performance. The experimental results of our evaluation using WDBench [2] show that our approach outperforms the state of the art.

As part of our future work, we plan to improve the variable selection process of the proposed algorithm by combining the strategies based on the reduction factor and minimum cardinality. We also plan to reduce the computational overhead of our default estimation by terminating the estimation once the active minimum cardinality has been exceeded. We have already used our approach for the evaluation of negated property sets and plan to further study their efficient evaluation. However, we have observed a lack of benchmark queries using negated property sets, which makes their optimization challenging.

Acknowledgments. This work has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070305 and the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW) within the project SAIL under the grant no NW21-059D.

References

1. Aimonier-Davat, J., Skaf-Molli, H., Molli, P., Dang, M.H., Nédelec, B.: Join ordering of SPARQL property path queries. In: Pesquita, C., Jiménez-Ruiz, E., McCusker, J.P., Faria, D., Dragoni, M., Dimou, A., Troncy, R., Hertling, S. (eds.) The Semantic Web - 20th International Conference, ESWC 2023, Hersonissos, Crete, Greece, May 28 - June 1, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13870, pp. 38–54. Springer (2023), https://doi.org/10.1007/978-3-031-33455-9_3
2. Angles, R., Aranda, C.B., Hogan, A., Rojas, C., Vrgoc, D.: Wdbench: A wikidata graph query benchmark. In: Sattler, U., Hogan, A., Keet, C.M., Presutti, V., Almeida, J.P.A., Takeda, H., Monnin, P., Pirrò, G., d'Amato, C. (eds.) The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23-27, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13489, pp. 714–731. Springer (2022), https://doi.org/10.1007/978-3-031-19433-7_41
3. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**(5), 68:1–68:40 (2017), https://doi.org/10.1145/3104031
4. Arroyuelo, D., Gómez-Brandón, A., Navarro, G.: Evaluating regular path queries on compressed adjacency matrices. In: Nardini, F.M., Pisanti, N., Venturini, R. (eds.) String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26-28, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14240, pp. 35–48. Springer (2023), https://doi.org/10.1007/978-3-031-43980-3_4
5. Arroyuelo, D., Hogan, A., Navarro, G., Reutter, J.L., Rojas-Ledesma, J., Soto, A.: Worst-case optimal graph joins in almost no space. In: Li, G., Li, Z., Idreos, S., Srivastava, D. (eds.) SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021. pp. 102–114. ACM (2021), https://doi.org/10.1145/3448016.3457256
6. Arroyuelo, D., Hogan, A., Navarro, G., Rojas-Ledesma, J.: Time- and space-efficient regular path queries. In: 38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022. pp. 3091–3105. IEEE (2022), https://doi.org/10.1109/ICDE53745.2022.00277
7. Atserias, A., Grohe, M., Marx, D.: Size bounds and query plans for relational joins. In: 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA. pp. 739–748. IEEE Computer Society (2008), https://doi.org/10.1109/FOCS.2008.43
8. Bigerl, A., Conrads, F., Behning, C., Sherif, M.A., Saleem, M., Ngomo, A.N.: Tentris - A tensor-based triple store. In: Pan, J.Z., Tamma, V.A.M., d'Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., Kagal, L. (eds.) The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12506, pp. 56–73. Springer (2020), https://doi.org/10.1007/978-3-030-62419-4_4
9. Bigerl, A., Conrads, L., Behning, C., Saleem, M., Ngomo, A.N.: Hashing the hypertrie: Space- and time-efficient indexing for SPARQL in tensors. In: Sattler, U., Hogan, A., Keet, C.M., Presutti, V., Almeida, J.P.A., Takeda, H., Monnin, P., Pirrò, G., d'Amato, C. (eds.) The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Virtual Event, October 23-27, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13489, pp. 57–73. Springer (2022), https://doi.org/10.1007/978-3-031-19433-7_4

10. Bonifati, A., Fletcher, G.H.L., Voigt, H., Yakovets, N.: Querying Graphs. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2018), https://doi.org/10.2200/S00873ED1V01Y201808DTM051
11. Bonifati, A., Martens, W., Timm, T.: Navigating the maze of wikidata query logs. In: Liu, L., White, R.W., Mantrach, A., Silvestri, F., McAuley, J.J., Baeza-Yates, R., Zia, L. (eds.) The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019. pp. 127–138. ACM (2019), https://doi.org/10.1145/3308558.3313472
12. Conrads, F., Lehmann, J., Saleem, M., Ngomo, A.N.: Benchmarking RDF storage solutions with IGUANA. In: Nikitina, N., Song, D., Fokoue, A., Haase, P. (eds.) Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks colocated with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017. CEUR Workshop Proceedings, vol. 1963. CEUR-WS.org (2017), https://ceur-ws.org/Vol-1963/paper621.pdf
13. Cucumides, T., Reutter, J.L., Vrgoc, D.: Size bounds and algorithms for conjunctive regular path queries. In: Geerts, F., Vandevoort, B. (eds.) 26th International Conference on Database Theory, ICDT 2023, March 28-31, 2023, Ioannina, Greece. LIPIcs, vol. 255, pp. 13:1–13:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023), https://doi.org/10.4230/LIPIcs.ICDT.2023.13
14. Demir, C., Ngomo, A.N.: Neuro-symbolic class expression learning. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China. pp. 3624–3632. ijcai.org (2023), https://doi.org/10.24963/ijcai.2023/403
15. Freitag, M.J., Bandle, M., Schmidt, T., Kemper, A., Neumann, T.: Adopting worst-case optimal joins in relational database systems. Proc. VLDB Endow. **13**(11), 1891–1904 (2020), http://www.vldb.org/pvldb/vol13/p1891-freitag.pdf
16. Gubichev, A., Bedathur, S.J., Seufert, S.: Sparqling kleene: fast property paths in RDF-3X. In: Boncz, P.A., Neumann, T. (eds.) First International Workshop on Graph Data Management Experiences and Systems, GRADES 2013, co-located with SIGMOD/PODS 2013, New York, NY, USA, June 24, 2013. p. 14. CWI/ACM (2013), http://event.cwi.nl/grades2013/14-gubichev.pdf
17. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013), retrieved 2023-21-11
18. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Neumaier, S., Ngomo, A.N., Polleres, A., Rashid, S.M., Rula, A., Schmelzeisen, L., Sequeda, J.F., Staab, S., Zimmermann, A.: Knowledge graphs. ACM Comput. Surv. pp. 71:1–71:37 (2021)
19. Hogan, A., Riveros, C., Rojas, C., Soto, A.: A worst-case optimal join algorithm for SPARQL. In: Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I.F., Hogan, A., Song, J., Lefrançois, M., Gandon, F. (eds.) The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11778, pp. 258–275. Springer (2019), https://doi.org/10.1007/978-3-030-30793-6_15
20. Kostylev, E.V., Reutter, J.L., Romero, M., Vrgoc, D.: SPARQL with property paths. In: Arenas, M., Corcho, Ó., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I. Lecture Notes in Com-

puter Science, vol. 9366, pp. 3–18. Springer (2015), https://doi.org/10.1007/978-3-319-25007-6_1

21. Malyshev, S., Krötzsch, M., González, L., Gonsior, J., Bielefeldt, A.: Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In: Vrandecic, D., Bontcheva, K., Suárez-Figueroa, M.C., Presutti, V., Celino, I., Sabou, M., Kaffee, L., Simperl, E. (eds.) The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11137, pp. 376–394. Springer (2018), https://doi.org/10.1007/978-3-030-00668-6_23
22. Ngo, H.Q., Porat, E., Ré, C., Rudra, A.: Worst-case optimal join algorithms. J. ACM **65**(3), 16:1–16:40 (2018), https://doi.org/10.1145/3180143
23. Ngo, H.Q., Ré, C., Rudra, A.: Skew strikes back: new developments in the theory of join algorithms. SIGMOD Rec. **42**(4), 5–16 (2013), https://doi.org/10.1145/2590989.2590991
24. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Trans. Database Syst. **34**(3), 16:1–16:45 (2009), https://doi.org/10.1145/1567274.1567278
25. Salas, J., Hogan, A.: Semantics and canonicalisation of SPARQL 1.1. Semantic Web **13**(5), 829–893 (2022), https://doi.org/10.3233/SW-212871
26. Schmidt, M., Meier, M., Lausen, G.: Foundations of SPARQL query optimization. In: Segoufin, L. (ed.) Database Theory - ICDT 2010, 13th International Conference, Lausanne, Switzerland, March 23-25, 2010, Proceedings. pp. 4–33. ACM International Conference Proceeding Series, ACM (2010), https://doi.org/10.1145/1804669.1804675
27. Syed, Z.H., Röder, M., Ngomo, A.N.: Unsupervised discovery of corroborative paths for fact validation. In: Ghidini, C., Hartig, O., Maleshkova, M., Svátek, V., Cruz, I.F., Hogan, A., Song, J., Lefrançois, M., Gandon, F. (eds.) The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11778, pp. 630–646. Springer (2019), https://doi.org/10.1007/978-3-030-30793-6_36
28. Vrgoč, D., Rojas, C., Angles, R., Arenas, M., Arroyuelo, D., Buil-Aranda, C., Hogan, A., Navarro, G., Riveros, C., Romero, J.: MillenniumDB: An Open-Source Graph Database System. Data Intelligence pp. 1–39 (06 2023), https://doi.org/10.1162/dint_a_00209
29. Wang, Y.R., Willsey, M., Suciu, D.: Free join: Unifying worst-case optimal and traditional joins. Proc. ACM Manag. Data **1**(2), 150:1–150:23 (2023), https://doi.org/10.1145/3589295
30. Yakovets, N., Godfrey, P., Gryz, J.: Query planning for evaluating SPARQL property paths. In: Özcan, F., Koutrika, G., Madden, S. (eds.) Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. pp. 1875–1889. ACM (2016). https://doi.org/10.1145/2882903.2882944, https://doi.org/10.1145/2882903.2882944