Acyclic Conjunctive Regular Path Queries are no Harder than Corresponding Conjunctive Queries

MAHMOUD ABO KHAMIS, RelationalAI, USA
ALEXANDRU-MIHAI HURJUI, University of Zurich, Switzerland
AHMET KARA, OTH Regensburg, Germany
DAN OLTEANU, University of Zurich, Switzerland
DAN SUCIU, University of Washington, USA

We present an output-sensitive algorithm for evaluating an acyclic Conjunctive Regular Path Query (CRPQ). Its complexity is written in terms of the input size, the output size, and a well-known parameter of the query that is called the "free-connex fractional hypertree width". Our algorithm improves upon the complexity of the recently introduced output-sensitive algorithm for acyclic CRPQs [4]. More notably, the complexity of our algorithm for a given acyclic CRPQ Q matches the best known output-sensitive complexity for the "corresponding" conjunctive query (CQ), that is the CQ that has the same structure as the CRPQ Q except that each RPQ is replaced with a binary atom (or a join of two binary atoms). This implies that it is not possible to improve upon our complexity for acyclic CRPQs without improving the state-of-the-art on output-sensitive evaluation for acyclic CQs. Our result is surprising because RPQs, and by extension CRPQs, are equivalent to recursive Datalog programs, which are generally poorly understood from a complexity standpoint. Yet, our result implies that the recursion aspect of acyclic CRPQs does not add any extra complexity on top of the corresponding (non-recursive) CQs, at least as far as output-sensitive analysis is concerned.

CCS Concepts: • **Information systems → Graph-based database models**; • **Theory of computation → Database query processing and optimization (theory)**; **Graph algorithms analysis**.

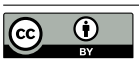
Additional Key Words and Phrases: graph databases; conjunctive regular path queries; output-sensitive algorithms; product graph

ACM Reference Format:
Mahmoud Abo Khamis, Alexandru-Mihai Hurjui, Ahmet Kara, Dan Olteanu, and Dan Suciu. 2026. Acyclic Conjunctive Regular Path Queries are no Harder than Corresponding Conjunctive Queries. *Proc. ACM Manag. Data* 4, 2 (PODS), Article 95 (May 2026), 24 pages. https://doi.org/10.1145/3801891

1 Introduction

Graph databases have become increasingly popular in recent years, with applications in various domains such as knowledge graphs, social networks, and the semantic web. These graph databases are often queried differently from traditional relational databases, using graph query languages such as SPARQL [23] and Cypher [21]. Recursion is at the heart of graph query languages, and it is often expressed using regular expressions over edge labels, leading to the notion of Regular Path Queries (RPQs) and their conjunctive extensions, Conjunctive Regular Path Queries (CRPQs). CRPQs are part of SPARQL, the W3C standard for querying RDF data, including commonly used

Authors' Contact Information: Mahmoud Abo Khamis, mahmoudabo@gmail.com, RelationalAI, Berkeley, CA, USA; Alexandru-Mihai Hurjui, hurjuialexandru12@gmail.com, University of Zurich, Zurich, Switzerland; Ahmet Kara, ahmet.kara@oth-regensburg.de, OTH Regensburg, Regensburg, Germany; Dan Olteanu, olteanu@ifi.uzh.ch, University of Zurich, Zurich, Switzerland; Dan Suciu, suciu@cs.washington.edu, University of Washington, Seattle, WA, USA.



ACM 2836-6573/2026/5-ART95
https://doi.org/10.1145/3801891

knowledge bases such as DBpedia and Wikidata [19, 27]. Recent analytical studies of SPARQL query logs indicate that the use of RPQs and CRPQs is on the rise [13]. For example, RPQs currently make up 38% of unique queries to Wikidata [14, 25].

In this paper, we study acyclic CRPQs in the data complexity setting, i.e., where the query is *fixed*. We provide an *output-sensitive* algorithm for evaluating acyclic CRPQs, i.e., an algorithm whose complexity depends on both the input size and output size. We show that our algorithm improves upon prior algorithms for acyclic CRPQs [4, 5]. We also show that our complexity matches the best known output-sensitive complexity for the corresponding acyclic conjunctive queries [24]. Hence, it is not possible to improve upon our complexity for acyclic CRPQs *without* a new breakthrough on output-sensitive evaluation of acyclic conjunctive queries.

Given an alphabet Σ, an *edge-labeled graph* $G = (V, E, \Sigma)$ is a directed graph with a vertex set V and an edge set E where each edge is labeled with a symbol from Σ. The size of G is $|V| + |E|$ and we denote it by N throughout the paper. Given an edge-labeled graph $G = (V, E, \Sigma)$, a *regular path query (RPQ)* is a regular expression over Σ. We denote a regular expression as $\overset{\leftrightarrow}{R}$. The output of an RPQ $\overset{\leftrightarrow}{R}$ on G is the set of all pairs of vertices $(v, u) \in V \times V$ where G contains a path from v to u whose label (i.e., the concatenation of the labels of its edges) belongs to the language defined by $\overset{\leftrightarrow}{R}$. Recent work [5] introduced an *output-sensitive* algorithm for evaluating RPQs, meaning that its complexity depends not just on the size N of the input graph but also on the output size OUT. Its complexity is $O(N + N \cdot \mathsf{OUT}^{1/2})$.[1]

Given an edge-labeled graph $G = (V, E, \Sigma)$, a *Conjunctive Regular Path Query (CRPQ)* Q is a conjunction of atoms followed by a projection onto a set of *free variables*, where each atom is by itself an RPQ. In particular, each atom has the form $\overset{\leftrightarrow}{R}(X, Y)$ where $\overset{\leftrightarrow}{R}$ is some regular expression over Σ and X and Y are two variables. For example, consider the following two CRPQs:

$$Q_1(X, Y_1, Y_2, Y_3) = \overset{\leftrightarrow}{R_1}(X, Y_1) \wedge \overset{\leftrightarrow}{R_2}(X, Y_2) \wedge \overset{\leftrightarrow}{R_3}(X, Y_3) \tag{1}$$

$$Q_2(Y_1, Y_2, Y_3) = \overset{\leftrightarrow}{R_1}(X, Y_1) \wedge \overset{\leftrightarrow}{R_2}(X, Y_2) \wedge \overset{\leftrightarrow}{R_3}(X, Y_3) \tag{2}$$

The first query Q_1 is asking for tuples of vertices (x, y_1, y_2, y_3) in G such that[2] for every $i \in [3]$, there is a path from x to y_i whose label belongs to the language defined by $\overset{\leftrightarrow}{R_i}$. The second query Q_2 is asking for triples of vertices (y_1, y_2, y_3) such that there exists a vertex x where for every $i \in [3]$, there is a path from x to y_i whose label belongs to the language defined by $\overset{\leftrightarrow}{R_i}$. In this paper, we are only interested in CRPQs that are *acyclic*, which means that the query structure is a forest.[3] Both Q_1 and Q_2 above are acyclic since they both have a query structure of a 3-star. Acyclic CRPQs can be *free-connex*, which is equivalent[4] to saying that the free variables are connected. This is true for Q_1 but not for Q_2 since, in the latter, the free variables Y_1, Y_2, Y_3 are not connected.

Recent work [4] has introduced an output-sensitive algorithm for evaluating acyclic CRPQs. At a high-level, you can think of their algorithm as "lifting" the Yannakakis algorithm [37] from acyclic conjunctive queries (CQs) to acyclic CRPQs. Let's take the query Q_1 above. We cannot obtain an output-sensitive algorithm for Q_1 by simply evaluating each RPQ $\overset{\leftrightarrow}{R_i}(X, Y_i)$ independently using the algorithm from [5] and then joining their outputs. This is because the output of each RPQ $\overset{\leftrightarrow}{R_i}(X, Y_i)$

[1]The complexity was stated in [5] as $O(N + N^{3/2} + \mathsf{OUT} \cdot N^{1/2})$. However, upon examining the analysis from [5], it turns out that the complexity can be expressed as $O(N + N \cdot \Delta + N \cdot \mathsf{OUT}/\Delta)$ for any $\Delta > 0$. To minimize this expression, we set $\Delta = \mathsf{OUT}^{1/2}$, yielding the stated complexity.

[2]For a natural number n, we use $[n]$ to denote $\{1, 2, \ldots, n\}$. When $n = 0$, $[n] = \emptyset$.

[3]We will see later that the notion of acyclicity for CRPQs is slightly different from acyclicity for CQs, in the sense that acyclic CRPQs cannot have *self-loops* or *parallel-edges*. See Definition 2.8 and Remark 2.9 for details.

[4]While this definition might seem different from the standard definition of free-connex acyclic CQs [11], the two definitions coincide for connected acyclic CRPQs because all atoms are binary. See Section 2.3.

could be much larger than the final output of Q_1 itself. Instead, the main insight from [4] is to start by "calibrating" the RPQs $\tilde{R}_i(X, Y_i)$ with each other before invoking the algorithm from [5] on them and joining their outputs. The calibration filters out vertices that don't contribute to the final output, similar to the calibration passes in the original Yannakakis algorithm [37]. Using this insight, the authors from [4] show how to evaluate any acyclic CRPQ that is *free-connex* in time $O(N + N \cdot \mathsf{OUT}^{1/2} + \mathsf{OUT})$, where OUT is the output size. This applies to the example query Q_1 above.

Extending the approach from [4] from free-connex CRPQs to acyclic CRPQs that are *not* free-connex turns out to be more challenging. Such non-free-connex queries are transformed in [4] into free-connex ones by "promoting" some non-free variables to become free. For example, Q_2 above can be transformed into a free-connex query by promoting X to become free, thus obtaining Q_1. Using this approach, the authors of [4] obtain a "weak" output-sensitive algorithm for Q_2 whose runtime is $O(N + N \cdot \mathsf{OUT}_1^{1/2} + \mathsf{OUT}_1)$ where OUT_1 is the output size of the *transformed query* Q_1, which could be much larger than the original output size OUT_2 of Q_2. To rewrite the runtime in terms of the original output size OUT_2, the authors use a weak upper bound of $\mathsf{OUT}_1 \leq \mathsf{OUT}_2 \cdot |V|$, which gives a runtime of $O(N + N \cdot \mathsf{OUT}_2^{1/2} \cdot |V|^{1/2} + \mathsf{OUT}_2 \cdot |V|)$. This is unsatisfactory because it introduces a new parameter $|V|$ into the runtime, is still a loose bound for many input instances, and doesn't address non-free-connex queries at a fundamental level. Instead, it over-approximates them using blackbox algorithms that were not designed for them. In the worst case, $|V|$ could be as large as N, making the runtime $O(N + N^{3/2} \cdot \mathsf{OUT}_2^{1/2} + N \cdot \mathsf{OUT}_2)$, which can be significantly improved as we explain below.

Instead, in this paper, we develop a custom-made approach for acyclic CRPQs, both free-connex and non-free-connex, where the concept of "free-connex" naturally occurs at the heart of our approach. Its complexity depends on N, OUT, and a parameter of the query Q, which has been known in the literature on CQs for many years under different names [6, 32], but was recently referred to as the *free-connex fractional hypertree width*, of the query Q [24], denoted $\mathsf{fn\text{-}fhtw}(Q)$.[5] The free-connex fractional hypertree width is a measure of how far a query is from being free-connex. It is traditionally defined for CQs, but we extend it to CRPQs in the natural way. Its value is 1 when the query is free-connex, and it increases as the query becomes "less" free-connex. For example, the query Q_1 above has $\mathsf{fn\text{-}fhtw}(Q_1) = 1$ since it is free-connex, while Q_2 has $\mathsf{fn\text{-}fhtw}(Q_2) = 3$. Our main result says that any acyclic CRPQ Q can be evaluated in the following time, in data complexity:

$$O(N + N \cdot \mathsf{OUT}^{1 - \frac{1}{\max(\mathsf{fn\text{-}fhtw}(Q), 2)}} + \mathsf{OUT}) \tag{3}$$

The above complexity collapses back to the complexity from [5] for RPQs and to the complexity from [4] for free-connex acyclic CRPQs. For non-free-connex acyclic CRPQs, our complexity is significantly better than the one from [4]. For example, our complexity for Q_2 becomes $O(N + N \cdot \mathsf{OUT}_2^{2/3} + \mathsf{OUT}_2)$, where OUT_2 is the output size of Q_2. This is a more natural runtime expression and is also *always*[6] upper bounded by the runtime from [4], which was $O(N + N^{3/2} \cdot \mathsf{OUT}_2^{1/2} + N \cdot \mathsf{OUT}_2)$. See Appendix C for a detailed comparison with [4].

But now we ask: How good is our runtime bound from Eq. (3)? Is there a way to establish a lower bound showing how optimal it is? To answer this question, we use conjunctive queries as "points of reference" to establish *lower bounds* on CRPQs. In particular, each CRPQ Q must be at least as hard to evaluate as the "corresponding" CQ Q', where each RPQ atom $\tilde{R}(X, Y)$ in Q is replaced

[5] In particular, the free-connex fractional hypertree width is just like the fractional hypertree width except that we only consider *free-connex* tree decompositions [11]. See Sec. 2.3.

[6] To compare the two runtimes, note that for Q_2, we always have $\mathsf{OUT}_2 \leq N^3$.

with a binary relation atom $S(X, Y)$ in Q' of size N. For example, the CRPQ Q_2 above is at least as hard to evaluate as the following CQ:

$$Q_2'(Y_1, Y_2, Y_3) = S_1(X, Y_1) \wedge S_2(X, Y_2) \wedge S_3(X, Y_3)$$

This is because given relation instances for S_1, S_2, S_3 for Q_2', we can construct a corresponding edge-labeled graph G over an alphabet $\Sigma = \{S_1, S_2, S_3\}$ where each tuple $S_i(x, y)$ corresponds to an edge from x to y labeled with S_i. And now, evaluating Q_2' reduces to evaluating Q_2 on G where each $\overset{\leftrightarrow}{R_i}$ is chosen to be a single symbol S_i. In fact, a CRPQ Q could be even harder. For example, even a single RPQ $\overset{\leftrightarrow}{R}$ is at least as hard to evaluate as the following k-path CQ, for every $k \geq 1$:

$$Q_{k\text{-path}}(X_1, X_{k+1}) = S_1(X_1, X_2) \wedge S_2(X_2, X_3) \wedge \cdots \wedge S_k(X_k, X_{k+1}) \tag{4}$$

This is because the regular expression $\overset{\leftrightarrow}{R}$ can be chosen to be the concatenation of k symbols $S_1 \cdots S_k$. For a given acyclic CQ Q', the best known output-sensitive *combinatorial algorithm*[7] has the following complexity [18, 24]:

$$O(N + N \cdot \mathsf{OUT}^{1 - \frac{1}{\text{fn-fhtw}(Q')}} + \mathsf{OUT}) \tag{5}$$

For example, for Q_2' above, the best known combinatorial algorithm runs in time $O(N + N \cdot \mathsf{OUT}^{2/3} + \mathsf{OUT})$, which happens to match the runtime of our algorithm for Q_2, which is also combinatorial. But this means that we cannot improve our complexity for the CRPQ Q_2 *without* improving the complexity for the CQ Q_2' from [9, 24]. More generally, our runtime expression for an acyclic CRPQ Q from Eq. (3) matches the runtime expression for the corresponding CQ Q' from Eq. (5), whenever the free-connex fractional hypertree width is at least two. Moreover, when the free-connex fractional hypertree width is one[8], our runtime expression becomes $O(N + N \cdot \mathsf{OUT}^{1/2} + \mathsf{OUT})$, which matches the runtime of the best known combinatorial algorithm[9] for the CQ $Q_{k\text{-path}}$ above for $k \geq 2$ [24]. This means that improving our runtime bound any further can *only* happen if a new breakthrough is made on the CQ side. In other words, our output-sensitive algorithm for acyclic CRPQs is the best possible, as far as we understand CQ evaluation today.

2 Preliminaries

2.1 Languages and Graphs

An *alphabet* Σ is a finite set of symbols. The set of all strings over an alphabet Σ is denoted by Σ^*. The set of all strings over Σ of length k is denoted by Σ^k. The empty string is denoted by ϵ. A *language* L over an alphabet Σ is a subset of Σ^*. Given a string $w \in \Sigma^*$, the *reverse* of w, denoted w^R, is defined recursively as follows: $w^R = \epsilon$ if $w = \epsilon$ and $w^R = \sigma \cdot v^R$ if $w = v \cdot \sigma$ for some $v \in \Sigma^*$ and $\sigma \in \Sigma$. The reverse of a language L is $L^R = \{w^R \mid w \in L\}$. We use the standard definitions of regular expressions and regular languages over an alphabet Σ. Throughout the paper, we denote regular expressions as $\overset{\leftrightarrow}{R}, \overset{\leftrightarrow}{S}$, etc. Given a regular expression $\overset{\leftrightarrow}{R}$, we use $L(\overset{\leftrightarrow}{R})$ to denote the language defined by $\overset{\leftrightarrow}{R}$.

Definition 2.1 (Edge-Labeled Graph)**.** An edge-labeled graph $G = (V, E, \Sigma)$ consists of a finite set V of vertices, a finite set Σ of edge labels, and a set $E \subseteq V \times \Sigma \times V$ of labeled directed edges. Every triple (v, σ, u) in E denotes an edge from vertex v to vertex u labeled with σ. The size of G is $|V| + |E|$, i.e., the total number of its vertices and edges.[10]

[7] *Combinatorial algorithms* is an under-defined concept in the algorithms literature that basically refers to algorithms that don't use fast matrix multiplication, e.g. Strassen's algorithm [34].

[8] The free-connex fractional hypertree width for acyclic queries takes only *integer* values [24]. See Proposition 2.16.

[9] In particular, when $k \geq 2$, we have $\text{fn-fhtw}(Q_{k\text{-path}}) = 2$, hence the runtime from Eq. (5) becomes $O(N + N \cdot \mathsf{OUT}^{1/2} + \mathsf{OUT})$.

[10] We neglect the size of Σ by assuming that it does not contain any symbol that does not appear as an edge label in G.

Definition 2.2 (A Path in an Edge-Labeled Graph)**.** Given an edge-labeled graph $G = (V, E, \Sigma)$ and two vertices $v, u \in V$, a *path* p from v to u of length k for some natural number k is a sequence of $k+1$ vertices $w_0 := v, w_1, \ldots, w_k := u$ and a sequence of k edge labels $\sigma_1, \sigma_2, \ldots, \sigma_k$ such that for every $i \in [k]$, there is an edge $(w_{i-1}, \sigma_i, w_i) \in E$. The *label* of the path p, denoted by $\sigma(p)$, is the string $\sigma_1\sigma_2 \ldots \sigma_k \in \Sigma^k$.

Definition 2.3 (Transpose of an Edge-Labeled Graph)**.** Given an edge-labeled graph $G = (V, E, \Sigma)$, let Σ^{-1} be a set of fresh symbols: one distinct fresh symbol σ^{-1} for each symbol $\sigma \in \Sigma$. The *transpose* of G is the edge-labeled graph $G^T = (V, E^T, \Sigma^{-1})$, where $E^T \stackrel{\text{def}}{=} \{(u, \sigma^{-1}, v) \mid (v, \sigma, u) \in E\}$.

Definition 2.4 (Inverse of a Regular Expression)**.** Given a regular expression $\overset{\leftrightsquigarrow}{R}$ over an alphabet Σ, let Σ^{-1} be a set of fresh symbols: one distinct fresh symbol σ^{-1} for each symbol $\sigma \in \Sigma$. The *inverse* of $\overset{\leftrightsquigarrow}{R}$, denoted $\overset{\leftrightsquigarrow}{R}^{-1}$, is a regular expression over Σ^{-1} that defines the reverse language of $L(\overset{\leftrightsquigarrow}{R})$ but where every symbol σ is replaced by σ^{-1}. In particular, $\overset{\leftrightsquigarrow}{R}^{-1}$ is defined inductively as follows:

$$\overset{\leftrightsquigarrow}{R}^{-1} \stackrel{\text{def}}{=} \begin{cases} \epsilon & \text{if } \overset{\leftrightsquigarrow}{R} = \epsilon, \\ \sigma^{-1} & \text{if } \overset{\leftrightsquigarrow}{R} = \sigma \in \Sigma, \\ (\overset{\leftrightsquigarrow}{R_1}^{-1} + \overset{\leftrightsquigarrow}{R_2}^{-1}) & \text{if } \overset{\leftrightsquigarrow}{R} = \overset{\leftrightsquigarrow}{R_1} + \overset{\leftrightsquigarrow}{R_2}, \\ (\overset{\leftrightsquigarrow}{R_2}^{-1} \cdot \overset{\leftrightsquigarrow}{R_1}^{-1}) & \text{if } \overset{\leftrightsquigarrow}{R} = \overset{\leftrightsquigarrow}{R_1} \cdot \overset{\leftrightsquigarrow}{R_2}, \\ (\overset{\leftrightsquigarrow}{R_1}^{-1})^* & \text{if } \overset{\leftrightsquigarrow}{R} = (\overset{\leftrightsquigarrow}{R_1})^*. \end{cases}$$

Definition 2.5 (Product Graph, $G_1 \times G_2$)**.** Given two edge-labeled graphs $G_1 = (V_1, E_1, \Sigma)$ and $G_2 = (V_2, E_2, \Sigma)$, the *product graph* $G_1 \times G_2$ is a directed graph $G = (V, E)$ (without edge labels) where $V \stackrel{\text{def}}{=} V_1 \times V_2$ and $E \stackrel{\text{def}}{=} \{((v_1, v_2), (u_1, u_2)) \mid \exists \sigma \in \Sigma : (v_1, \sigma, u_1) \in E_1 \wedge (v_2, \sigma, u_2) \in E_2\}$.

2.2 Queries

A Conjunctive Query (CQ) has the form $Q(\boldsymbol{F}) = R_1(\boldsymbol{X}_1) \wedge \cdots \wedge R_n(\boldsymbol{X}_n)$, where $\mathsf{vars}(Q) \stackrel{\text{def}}{=} \bigcup_{i \in [n]} \boldsymbol{X}_i$ is the set of variables, $\mathsf{free}(Q) \stackrel{\text{def}}{=} \boldsymbol{F} \subseteq \mathsf{vars}(Q)$ is the set of *free* variables, $\mathsf{bound}(Q) \stackrel{\text{def}}{=} \mathsf{vars}(Q) \setminus \mathsf{free}(Q)$ is the set of *bound* variables, and $\mathsf{atoms}(Q)$ is the set of atoms $R_i(\boldsymbol{X}_i)$. We follow the standard semantics for CQs. The query graph of a CQ Q is a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} := \mathsf{vars}(Q)$ and $\mathcal{E} := \{\boldsymbol{X}_i \mid R_i(\boldsymbol{X}_i) \in \mathsf{atoms}(Q)\}$. We call a CQ Q *acyclic* if its query graph is α-acyclic, which means that we can construct a tree whose nodes are the sets $\boldsymbol{X}_i$ for $i \in [n]$ where each variable $X \in \boldsymbol{V}$ appears in a connected subtree.

Definition 2.6 (Conjunctive Regular Path Queries)**.** A *Conjunctive Regular Path Query (CRPQ)* Q over an alphabet Σ is of the form $Q(\boldsymbol{F}) = \overset{\leftrightsquigarrow}{R_1}(X_1, Y_1) \wedge \cdots \wedge \overset{\leftrightsquigarrow}{R_n}(X_n, Y_n)$, where: each $\overset{\leftrightsquigarrow}{R_i}$ is a regular expression over Σ; each $\overset{\leftrightsquigarrow}{R_i}(X_i, Y_i)$ is an *atom*; $\mathsf{vars}(Q) \stackrel{\text{def}}{=} \bigcup_{i \in [n]} \{X_i, Y_i\}$ is the set of variables; $\mathsf{free}(Q) \stackrel{\text{def}}{=} \boldsymbol{F} \subseteq \mathsf{vars}(Q)$ is the set of *free* variables; $\mathsf{bound}(Q) \stackrel{\text{def}}{=} \mathsf{vars}(Q) \setminus \mathsf{free}(Q)$ is the set of *bound* variables; and $\mathsf{atoms}(Q)$ is the set of atoms. The answer to the CRPQ Q on an input edge-labeled graph $G = (V, E, \Sigma)$ is defined as follows: A mapping $\mu : \mathsf{free}(Q) \to V$ is in the answer of Q if it can be extended to a mapping $\mu' : \mathsf{free}(Q) \cup \mathsf{bound}(Q) \to V$ such that G has a path from $\mu'(X_i)$ to $\mu'(Y_i)$ labeled by a string from $L(\overset{\leftrightsquigarrow}{R_i})$, for every $i \in [n]$. We represent a mapping μ by the tuple $(\mu(X))_{X \in \mathsf{free}(Q)}$, assuming a fixed order on the free variables.

Query Graphs and Acyclic CRPQs. In order to define the query graph of a CRPQ, we first recall the definition of undirected multigraphs. An *undirected multigraph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a finite set of vertices $\mathcal{V}$ and a multiset of edges $\mathcal{E}$ such that each edge is a subset of $\mathcal{V}$ of size two or

one. An edge of size one is called a *self-loop*. The *degree* of a vertex v in $\mathcal{G}$ is the number of edges incident to v. A vertex that has degree one or zero is called a *leaf*. Given two vertices $v, u \in \mathcal{V}$, a *path* from v to u in $\mathcal{G}$ is a sequence of vertices $w_0 := v, w_1, \ldots, w_k := u$ for some $k \geq 1$ such that for every $i \in [k]$, there is an edge $\{w_{i-1}, w_i\} \in \mathcal{E}$.

Definition 2.7 (Query Graph of a CRPQ)**.** The *query graph* of a CRPQ Q is an undirected multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} := \mathsf{vars}(Q)$ and edge multiset $\mathcal{E} := \{\!\{\{X, Y\} \mid \overset{\leftrightarrow}{R}(X, Y) \in \mathsf{atoms}(Q)\}\!\}$.

A *cycle* in an undirected multigraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a sequence of the form $v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1} := v_1$ of k distinct vertices $v_1, \ldots, v_k \in \mathcal{V}$ and k distinct edges $e_1, \ldots, e_k \in \mathcal{E}$ such that $k \geq 1$ and for every $i \in [k]$, we have $e_i = \{v_i, v_{i+1}\}$. A multigraph is called *acyclic* if it has no cycles.

Definition 2.8 (Acyclic CRPQs)**.** A CRPQ Q is called *acyclic* if its query graph is acyclic.

Remark 2.9 (Distinction between acyclicity in CRPQs versus CQs)**.** Note that according to the above definition, an acyclic CRPQ cannot have a *self-loop*, i.e., an atom of the form $\overset{\leftrightarrow}{R}(X, X)$, or *parallel edges*, i.e., two atoms sharing the same variables. This is more restricted than the definition of (α-)acyclicity for CQs in the literature, where a CQ with an atom $R(X, X)$ or two atoms $R_1(X, Y), R_2(X, Y)$ can still be acyclic. However, this restriction is unavoidable because a CRPQ with a self-loop, $\overset{\leftrightarrow}{R}(X, X)$, is already as hard as evaluating the following k-cycle conjunctive query for any $k \geq 3$:

$$Q_{k\text{-cycle}}(X_1) = S_1(X_1, X_2) \wedge S_2(X_2, X_3) \wedge \cdots \wedge S_k(X_k, X_1) \tag{6}$$

This is because the regular expression $\overset{\leftrightarrow}{R}$ could be chosen to be a concatenation of k symbols, $S_1 \cdots S_k$. A similar reasoning applies to parallel edges, where a CRPQ consisting of only two parallel edges is already as hard as $Q_{k\text{-cycle}}$ for any $k \geq 3$.

In light of the above discussion it is possible to define acyclicity for CRPQs in terms of their *k-expansions* to CQs:

Definition 2.10 (k-expansion of a CRPQ)**.** Given a CRPQ Q and a natural number $k \geq 1$, the *k-expansion* of Q, is a CQ Q'_k that is obtained from Q by replacing every atom $\overset{\leftrightarrow}{R}(X, Y)$ in Q with a k-path of binary atoms $S(X, Z_1), S(Z_1, Z_2), \ldots, S(Z_{k-1}, Y)$, where $Z_1, \ldots, Z_{k-1}$ are *fresh* variables and S is an arbitrary relation symbol.

Proposition 2.11. *A CRPQ Q is acyclic if and only if its 3-expansion is an acyclic CQ.*

The reason we use 3-expansions above (and not just 2-expansions) is because a self-loop will only result in a cyclic CQ after a 3-expansion.

A CRPQ Q is called *connected* if its query graph is connected. A variable in Q is called a *leaf* if it is a leaf in the query graph of Q. A query Q' is a subquery of Q if $\mathsf{atoms}(Q') \subseteq \mathsf{atoms}(Q)$ and $\mathsf{free}(Q') = \mathsf{free}(Q) \cap \mathsf{vars}(Q')$.

We state some basic observations on evaluating RPQs and CRPQs.

Proposition 2.12. *(The CRPQ $Q(X) = \overset{\leftrightarrow}{R}(X, Y)$ is evaluatable in linear time.) Given an edge-labeled graph of size N, the CRPQ $Q(X) = \overset{\leftrightarrow}{R}(X, Y)$ can be evaluated in $O(N)$ time.*

The following proposition follows immediately from Definitions 2.3 and 2.4. It basically says that to produce the output of an RPQ $\overset{\leftrightarrow}{R}(Y, X)$ where X and Y are swapped, we can simply take the *inverse* of $\overset{\leftrightarrow}{R}$ and evaluate it on the *transpose graph*.

Proposition 2.13. *(The CRPQ $Q(X,Y) = \overset{\rightsquigarrow}{R}(Y,X)$ is equivalent to an RPQ $\overset{\rightsquigarrow}{R}^{-1}(X,Y)$.) The answer of the CRPQ $Q(X,Y) = \overset{\rightsquigarrow}{R}(Y,X)$ on an edge-labeled graph G is the same as the answer of the RPQ $\overset{\rightsquigarrow}{R}^{-1}(X,Y)$ on the transpose graph G^T.*

Proposition 2.14. *(The query $Q(X,Y) = \overset{\rightsquigarrow}{R}(X,Y) \wedge S(Y)$ is equivalent to an RPQ.) Let $\overset{\rightsquigarrow}{R}(X,Y)$ be an RPQ over an edge-labeled graph $G = (V,E,\Sigma)$, and $S \subseteq V$ be a set of vertices viewed as a unary relation. The answer of the query $Q(X,Y) = \overset{\rightsquigarrow}{R}(X,Y) \wedge S(Y)$ on G is the same as the answer of an RPQ $\overset{\rightsquigarrow}{R}'(X,Y)$ on an edge-labeled graph $G' = (V,E',\Sigma')$ that can be constructed from G in linear time.*

2.3 Width Measures

The following width measures were originally defined for conjunctive queries (CQs), but they extend naturally to CRPQs.

Given a CRPQ (or CQ) Q, a *tree decomposition* of Q is a pair (T,χ), where T is a tree and $\chi : \mathsf{nodes}(T) \to 2^{\mathsf{vars}(Q)}$ is a map from the nodes of T to subsets of $\mathsf{vars}(Q)$ that satisfies the following properties:

- For every atom $\overset{\rightsquigarrow}{R}(X,Y) \in \mathsf{atoms}(Q)$, there is a node $t \in \mathsf{nodes}(T)$ such that $X, Y \in \chi(t)$. (In case of a CQ Q, for every atom $R(\boldsymbol{X}) \in \mathsf{atoms}(Q)$, there is a node $t \in \mathsf{nodes}(T)$ such that $\boldsymbol{X} \subseteq \chi(t)$.)
- For every variable $X \in \mathsf{vars}(Q)$, the set $\{t \mid X \in \chi(t)\}$ forms a connected sub-tree of T.

Each set $\chi(t)$ is called a *bag* of the tree decomposition. Let $\mathsf{TD}(Q)$ denote the set of all tree decompositions of Q. A tree decomposition (T,χ) of Q is called *free-connex* [11, 33] if it contains a (possibly empty) connected subtree T_f with nodes $\mathsf{nodes}(T_f)$ that contains all and only the free variables of Q, i.e., $\bigcup_{t \in \mathsf{nodes}(T_f)} \chi(t) = \mathsf{free}(Q)$. Let $\mathsf{FTD}(Q)$ denote the set of all free-connex tree decompositions of Q. Note that if the query is *Boolean* (i.e. $\mathsf{free}(Q) = \emptyset$) or *full* (i.e. $\mathsf{free}(Q) = \mathsf{vars}(Q)$), then every tree decomposition of Q is free-connex, in which case $\mathsf{FTD}(Q) = \mathsf{TD}(Q)$.

Given a multigraph (or hypergraph) $\mathcal{G} = (\mathcal{V},\mathcal{E})$, and a set of vertices $Z \subseteq \mathcal{V}$, the *fractional edge cover number* of Z, $\rho^*_{\mathcal{G}}(Z)$, is defined as the optimal value of the following linear program:

$$\text{minimize} \sum_{e \in \mathcal{E}} x_e \qquad \text{subject to} \sum_{e : v \in e} x_e \geq 1, \quad \forall v \in Z \qquad \text{and} \qquad x_e \geq 0, \quad \forall e \in \mathcal{E} \tag{7}$$

Given a CRPQ (or CQ) Q and a set of variables $Z \subseteq \mathsf{vars}(Q)$, we use $\rho^*_Q(Z)$ to denote $\rho^*_{\mathcal{G}}(Z)$ where $\mathcal{G}$ is the query graph of Q.

Definition 2.15 (Free-Connex Fractional Hypertree Width)**.** Given a CRPQ (or CQ) Q, the *fractional hypertree width*, and the *free-connex fractional hypertree width* of Q, are defined respectively as follows:

$$\mathsf{fhtw}(Q) \overset{\text{def}}{=} \min_{(T,\chi) \in \mathsf{TD}(Q)} \max_{t \in \mathsf{nodes}(T)} \rho^*_Q(\chi(t)), \tag{8}$$

$$\text{fn-fhtw}(Q) \overset{\text{def}}{=} \min_{(T,\chi) \in \mathsf{FTD}(Q)} \max_{t \in \mathsf{nodes}(T)} \rho^*_Q(\chi(t)). \tag{9}$$

We state below some basic properties of the free-connex fractional hypertree width. Similar to $\rho^*_Q(Z)$ (and $\rho^*_{\mathcal{G}}(Z)$) defined above, we define the *integral* edge cover number $\rho_Q(Z)$ (and $\rho_{\mathcal{G}}(Z)$) by replacing the constraint $x_e \geq 0$ in the linear program (7) with $x_e \in \{0,1\}$, thus making it an integer linear program. We define the *generalized hypertree width* $\mathsf{ghtw}(Q)$, and the *free-connex generalized hypertree width* $\text{fn-ghtw}(Q)$, similar to Equations (8) and (9) respectively, where we replace ρ^*_Q with ρ_Q. An acyclic CRPQ (or acyclic CQ) is *free-connex* if it has a free-connex tree decomposition where for every bag $\chi(t)$, there is an atom containing all variables in $\chi(t)$. The following properties

of the free-connex fractional hypertree width are folklore and have been proved in various forms in the literature, e.g., [6, 11, 24, 32].

Proposition 2.16. *For any acyclic CRPQ (or acyclic CQ) Q, the following claims hold:*

- *fn-fhtw(Q) is an integer. In particular, fn-fhtw(Q) = fn-ghtw(Q).*
- *fn-fhtw(Q) is 1 if and only if Q is free-connex.*

Definition 2.17 (Trivial CRPQs)**.** An acyclic CRPQ Q is called *trivial* if each connected subquery of Q has at most one free variable.

Trivial acyclic CRPQs are known to be solvable in time $O(N + \mathsf{OUT})$ [4].

Proposition 2.18 (fn-fhtw of k-expansion). *Given an acyclic CRPQ Q and a natural number $k \geq 2$, let Q'_k be the k-expansion of Q (Definition 2.10). The following holds:*

$$\textit{fn-fhtw}(Q'_k) \quad = \quad \begin{cases} 1 & \text{if } Q \text{ is trivial,} \\ \max(\textit{fn-fhtw}(Q), 2) & \text{if } Q \text{ is non-trivial.} \end{cases} \tag{10}$$

As an example, the CRPQ $Q(X, Y) = \overset{*}{\overset{*}{R}}(X, Y)$ has $\mathsf{fn\text{-}fhtw}(Q) = 1$, but its k-expansion is $Q_{k\text{-path}}$ (Eq. (4)), which has $\mathsf{fn\text{-}fhtw}(Q_{k\text{-path}}) = 2$ for all $k \geq 2$.

2.4 Data Statistics

Given a set of variables $\boldsymbol{X}$, we refer to a tuple $\boldsymbol{x}$ over $\boldsymbol{X}$ as an $\boldsymbol{X}$*-tuple.*

Definition 2.19 (Degrees in a relation)**.** Let $R(\boldsymbol{Z})$ be a relation over a variable set $\boldsymbol{Z}$, and $\boldsymbol{X}, \boldsymbol{Y} \subseteq \boldsymbol{Z}$ be two (disjoint) subsets. Given an $\boldsymbol{X}$-tuple $\boldsymbol{x}$, the *degree of $\boldsymbol{Y}$ given $\boldsymbol{X} = \boldsymbol{x}$ in R*, denoted by $\deg_R(\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x})$, is the number of different $\boldsymbol{Y}$-tuples $\boldsymbol{y}$ that appear in R together with $\boldsymbol{x}$. The *degree of $\boldsymbol{Y}$ given $\boldsymbol{X}$ in R*, denoted by $\deg_R(\boldsymbol{Y}|\boldsymbol{X})$, is the maximum degree of $\boldsymbol{Y}$ given $\boldsymbol{X} = \boldsymbol{x}$ in R over all $\boldsymbol{X}$-tuples $\boldsymbol{x}$:

$$\begin{aligned} \deg_R(\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x}) &\stackrel{\text{def}}{=} |\pi_{\boldsymbol{Y}}(\sigma_{\boldsymbol{X}=\boldsymbol{x}}(R))|, \\ \deg_R(\boldsymbol{Y}|\boldsymbol{X}) &\stackrel{\text{def}}{=} \max_{\boldsymbol{x} \in \pi_{\boldsymbol{X}}(R)} \deg_R(\boldsymbol{Y}|\boldsymbol{X} = \boldsymbol{x}). \end{aligned}$$

3 Results

The following theorem is our main result:

Theorem 3.1. Let Q be any acyclic CRPQ, and G be an input edge-labeled graph of size N. Then, Q over G can be answered in time $O(N + N \cdot \mathsf{OUT}^{1 - \frac{1}{\max(\mathsf{fn\text{-}fhtw}(Q), 2)}} + \mathsf{OUT})$ in data complexity, where OUT is the output size, and $\mathsf{fn\text{-}fhtw}(Q)$ is the free-connex fractional hypertree width of Q (Definition 2.15).

Our algorithm for acyclic CRPQs is *combinatorial*, i.e., it does not use fast matrix multiplication. In order to assess how good its runtime is, we compare it below to the best known runtime for acyclic CQs over all combinatorial algorithms [18, 24].

Theorem 3.2 ([24])**.** Let Q be any acyclic CQ and D be an input database instance of size N. Then, Q over D can be answered in time $O(N + N \cdot \mathsf{OUT}^{1 - \frac{1}{\mathsf{fn\text{-}fhtw}(Q)}} + \mathsf{OUT})$ in data complexity, where OUT is the output size, and $\mathsf{fn\text{-}fhtw}(Q)$ is the free-connex fractional hypertree width of Q.

There are two differences between the above two theorems (besides the obvious fact that one is for CRPQs while the other is for CQs). The first (subtle) difference lies in the definition of acyclicity for CRPQs versus CQs. See Remark 2.9 for a discussion. The other difference is in the runtime expression, where in the CRPQ case, we take the maximum of fn-fhtw(Q) and 2. Taking this maximum is unavoidable by Proposition 2.18. This is because solving a CRPQ Q is at least as hard as solving its k-expansion Q'_k (which is a CQ) for every $k \geq 2$. Excluding *trivial*[11] CRPQs, Proposition 2.18 says fn-fhtw(Q'_k) = max(fn-fhtw(Q), 2). This shows that it is not possible to improve upon our runtime for acyclic CRPQs *without* improving the runtimes of the state-of-the-art combinatorial algorithms for acyclic CQs. In other words, our result links the fate of output-sensitive evaluation of acyclic CRPQs to that of acyclic CQs.

We now prove our Theorem 3.1. To that end, we will break it down into two steps: In the first step, we reduce the problem of answering an acyclic CRPQ to answering "free-leaf" CRPQs (Lemma 3.4). In the second step, we show how to answer free-leaf CRPQs efficiently (Lemma 3.5).

Definition 3.3 (A free-leaf CRPQ)**.** A CRPQ Q is called *free-leaf* if it is acyclic, connected, and has a set of free variables which is exactly the leaves of its query graph.

Lemma 3.4 (Reduction from an acyclic CRPQ to free-leaf CRPQs). *Let Q be any acyclic CRPQ, and G be an input edge-labeled graph of size N. Then, in $O(N)$-time in data complexity, we can construct k free-leaf CRPQs $Q_1, \ldots, Q_k$ and k corresponding edge-labeled graphs $G_1, \ldots, G_k$ for some constant k, satisfying the following:*

- $|free(Q_i)| \leq \max(fn\text{-}fhtw(Q), 2)$*, for each* $i \in [k]$.
- *The output size of each Q_i over G_i is at most OUT, where OUT is the output size of Q over G.*
- *The answer of Q over G can be computed from the answers of $Q_1, \ldots, Q_k$ over $G_1, \ldots, G_k$ in time $O(OUT)$.*

Lemma 3.5 (Solving free-leaf CRPQs). *Let Q be any free-leaf CRPQ, and G be an input edge-labeled graph of size N. Then, Q can be answered in time $O(N + N \cdot OUT^{1-\frac{1}{|free(Q)|}})$ in data complexity, where OUT is the output size.*

3.1 Proof of Lemma 3.4

As a running example in this section, we will use the following acyclic CRPQ Q, which is depicted in Figure 1 (top-left):

$$\begin{aligned} Q(D,E,F,G,K,L) = &\overset{\curvearrowright}{R}_1(A,B) \wedge \overset{\curvearrowright}{R}_2(A,C) \wedge \overset{\curvearrowright}{R}_3(A,D) \wedge \overset{\curvearrowright}{R}_4(C,E) \wedge \overset{\curvearrowright}{R}_5(C,F) \wedge \overset{\curvearrowright}{R}_6(D,G) \wedge \\ &\overset{\curvearrowright}{R}_7(E,H) \wedge \overset{\curvearrowright}{R}_8(F,I) \wedge \overset{\curvearrowright}{R}_9(F,J) \wedge \overset{\curvearrowright}{R}_{10}(H,K) \wedge \overset{\curvearrowright}{R}_{11}(H,L) \end{aligned} \tag{11}$$

First, we introduce some concepts.

Definition 3.6 (Bound-connected CRPQ)**.** A CRPQ (or CQ) Q is called *bound-connected* if every pair of variables in vars(Q) is connected in the query graph of Q by a path that goes through only bound variables, bound(Q).

Note that every free-leaf CRPQ is bound-connected, but not every acyclic bound-connected CRPQ is free-leaf. Nevertheless, every acyclic bound-connected CRPQ can be made free-leaf in linear time, as we will see later (Prop. 3.10).

[11] See Definition 2.17. Trivial CRPQs are known to be solvable in $O(N + \text{OUT})$ time [4].

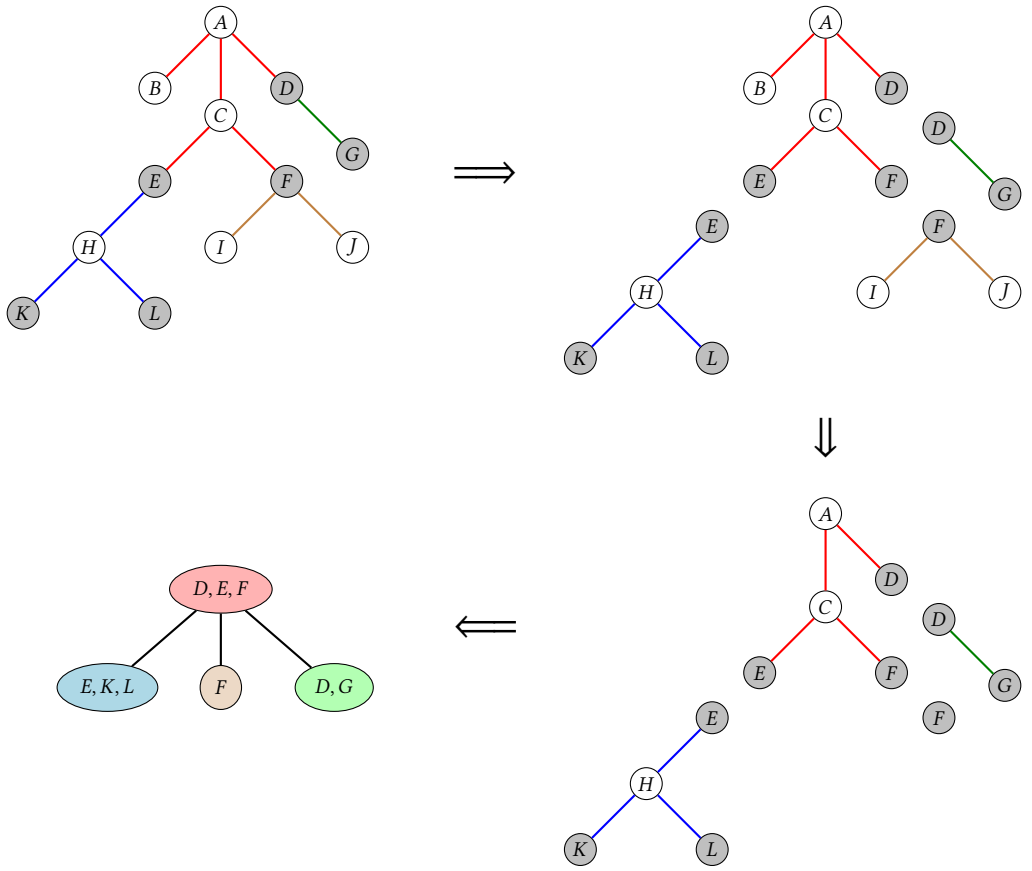


Fig. 1. An example demonstrating the reduction in Lemma 3.4. Top-left: The input acyclic CRPQ Q from Eq. (11), where free variables are shaded. Top-right: The bound connected components of Q (Definition 3.6). Bottom-right: The corresponding free-leaf CRPQs obtained by applying Proposition 3.10 to each component. Each free-leaf CRPQ is solved using the algorithm from Lemma 3.5, and the answers are stitched together into an acyclic full conjunctive query over only the free variables of Q. This query is depicted at the bottom-left, and it can be solved using Yannakakis algorithm.

Definition 3.7 (Bound-connected component of a CRPQ)**.** A *bound-connected component* of a CRPQ (or CQ) Q is a *maximal* bound-connected subquery Q' of Q, i.e., a bound-connected subquery Q' such that any subquery Q'' of Q with $\text{atoms}(Q'') \supset \text{atoms}(Q')$ is not bound-connected.

Figure 1 (top-right) depicts the bound-connected components of Q from Eq. (11), which are:

$$Q_1(D,E,F) = \overset{\leftrightarrow}{R}_1(A,B) \wedge \overset{\leftrightarrow}{R}_2(A,C) \wedge \overset{\leftrightarrow}{R}_3(A,D) \wedge \overset{\leftrightarrow}{R}_4(C,E) \wedge \overset{\leftrightarrow}{R}_5(C,F) \tag{12}$$

$$Q_2(D,G) = \overset{\leftrightarrow}{R}_6(D,G) \tag{13}$$

$$Q_3(E,K,L) = \overset{\leftrightarrow}{R}_7(E,H) \wedge \overset{\leftrightarrow}{R}_{10}(H,K) \wedge \overset{\leftrightarrow}{R}_{11}(H,L) \tag{14}$$

$$Q_4(F) = \overset{\leftrightarrow}{R}_8(F,I) \wedge \overset{\leftrightarrow}{R}_9(F,J) \tag{15}$$

The following statement was proven for conjunctive queries in prior work (Lemmas 3.3 and 3.4 in [24]). We give a simplified proof adapted to CRPQs and using variable elimination orders [2, 6].

Lemma 3.8 ([24]). *Let Q be an acyclic CRPQ (or acyclic CQ) and $Q_1, \ldots, Q_k$ its bound-connected components.*[12] *Then, we have*

$$\textit{fn-fhtw}(Q) = \max(\max_{i\in[k]} \rho^*_{Q_i}(\textit{free}(Q_i)), 1) \tag{16}$$

Proposition 3.9. *For any acyclic bound-connected CRPQ Q, $|\textit{free}(Q)| \le \max(\rho^*_Q(\textit{free}(Q)), 2)$.*

Lemma 3.8 and Prop. 3.9 imply that for any bound-connected component Q_i of an acyclic CRPQ Q, we must have

$$|\mathsf{free}(Q_i)| \quad \le \quad \max(\mathsf{fn\text{-}fhtw}(Q), 2) \tag{17}$$

Consider for example Q depicted in Figure 1. The free-connex fractional hypertree width of Q is $\mathsf{fn\text{-}fhtw}(Q) = 3$, whereas the components Q_1, Q_2, Q_3, Q_4 from Eqs. (12)–(15) have $3, 2, 3, 1$ free variables, respectively.

Proposition 3.10. *(Every acyclic bound-connected CRPQ can be transformed into free-leaf.) For any acyclic bound-connected CRPQ Q and input edge-labeled graph $G = (V, E, \Sigma)$ of size N, we can construct a free-leaf CRPQ Q' and an edge-labeled graph $G' = (V, E', \Sigma')$ in time $O(N)$ in data complexity such that $\textit{free}(Q) = \textit{free}(Q')$ and the answer of Q over G is the same as the answer of Q' over G'.*

The above proposition can be proved by repeatedly removing non-free leaf variables from Q using Propositions 2.12 and 2.14. Figure 1 (bottom-right) depicts the free-leaf CRPQs obtained by applying Prop. 3.10 to each bound-connected component of Q from Eq. (11).

Our reduction in Lemma 3.4 relies on taking the bound-connected components of Q, and applying Prop. 3.10 to each component. But there is a little twist: Consider for example the component Q_1 from Eq. (12). The output size of Q_1 could potentially be larger than the output size of Q, thus violating the second bullet point in Lemma 3.4. This is because, for example, the regular path query $\overset{\rightsquigarrow}{R_6}(D, G)$ could produce an empty output, but $\overset{\rightsquigarrow}{R_6}(D, G)$ is not part of Q_1. To resolve this issue, we have to modify Q_1 a bit. In particular, we define the following CRPQ, Q_D, whose body is the same as Q but has only one free variable, D:

$$\begin{aligned} Q_D(D) = &\overset{\rightsquigarrow}{R_1}(A, B) \wedge \overset{\rightsquigarrow}{R_2}(A, C) \wedge \overset{\rightsquigarrow}{R_3}(A, D) \wedge \overset{\rightsquigarrow}{R_4}(C, E) \wedge \overset{\rightsquigarrow}{R_5}(C, F) \wedge \overset{\rightsquigarrow}{R_6}(D, G) \wedge \\ &\overset{\rightsquigarrow}{R_7}(E, H) \wedge \overset{\rightsquigarrow}{R_8}(F, I) \wedge \overset{\rightsquigarrow}{R_9}(F, J) \wedge \overset{\rightsquigarrow}{R_{10}}(H, K) \wedge \overset{\rightsquigarrow}{R_{11}}(H, L) \end{aligned}$$

Similarly, we define a CRPQ Q_X for every free variable X of Q. Each such CRPQ Q_X can be answered in linear time by repeatedly applying Propositions 2.12 and 2.14 to eliminate leaf variables that are not X, as the following proposition shows.

Proposition 3.11 ([4]). *Every acyclic CRPQ Q with at most one free variable can be answered in time $O(N)$ in data complexity, where N is the size of the input edge-labeled graph.*

After evaluating each Q_X, we extend Q_1 by adding the outputs of the CRPQs Q_D, Q_E, and Q_F as extra filters to the body of Q_1:

$$Q'_1(D, E, F) = \overset{\rightsquigarrow}{R_1}(A, B) \wedge \overset{\rightsquigarrow}{R_2}(A, C) \wedge \overset{\rightsquigarrow}{R_3}(A, D) \wedge \overset{\rightsquigarrow}{R_4}(C, E) \wedge \overset{\rightsquigarrow}{R_5}(C, F) \wedge Q_D(D) \wedge Q_E(E) \wedge Q_F(F)$$

Now, the output of Q'_1 above is guaranteed to be the projection of the output of Q on the variables (D, E, F). By Proposition 2.14, Q'_1 above is still equivalent to a CRPQ:

$$Q'_1(D, E, F) = \overset{\rightsquigarrow}{R_1}(A, B) \wedge \overset{\rightsquigarrow}{R_2}(A, C) \wedge \overset{\rightsquigarrow}{R'_3}(A, D) \wedge \overset{\rightsquigarrow}{R'_4}(C, E) \wedge \overset{\rightsquigarrow}{R'_5}(C, F)$$

[12] The bound-connected components of an acyclic CRPQ are also acyclic.

The above CRPQ is acyclic bound-connected but not yet free-leaf because B is a leaf variable and it is not free. By applying Proposition 2.12, followed by Proposition 2.14, we can eliminate B, thus making Q_1' free-leaf:

$$Q_1'(D,E,F) = \overset{\leftrightsquigarrow}{R}_2'(A,C) \wedge \overset{\leftrightsquigarrow}{R}_3'(A,D) \wedge \overset{\leftrightsquigarrow}{R}_4'(C,E) \wedge \overset{\leftrightsquigarrow}{R}_5'(C,F)$$

Because it is free-leaf, the above query can be solved using the algorithm from Lemma 3.5 in time $O(N + N \cdot \mathsf{OUT}_1^{1-\frac{1}{|\mathsf{free}(Q_1')|}})$, where OUT_1 is the output size of Q_1', which satisfies $\mathsf{OUT}_1 \le \mathsf{OUT}$ by construction. By Eq. (17), this is bounded by $O(N + N \cdot \mathsf{OUT}^{1-\frac{1}{\max(\mathsf{fn\text{-}fhtw}(Q),2)}})$, as desired.

Similarly, we can define free-leaf CRPQs $Q_2'(D,G), Q_3'(E,K,L), Q_4'(F)$ and solve them in similar complexity. Finally, the answers from all components Q_i' form an acyclic full conjunctive query, which is depicted in Figure 1 (bottom-left). We can solve it using Yannakakis algorithm [37] in time $O(\sum_i \mathsf{OUT}_i + \mathsf{OUT}) = O(\mathsf{OUT})$.

3.2 Proof of Lemma 3.5

In this section, we demonstrate our proof of Lemma 3.5. First, we introduce a definition and some auxiliary propositions.

Definition 3.12 ((X,Δ)-restriction)**.** Let $R(X,\boldsymbol{Y})$ be a relation over variables $\{X\} \cup \boldsymbol{Y}$, where X is a single variable and $\boldsymbol{Y}$ is the set of remaining variables of R. Let Δ be a natural number. A relation $\overline{R}(X,\boldsymbol{Y})$ is called an (X,Δ)-*restriction* of $R(X,\boldsymbol{Y})$ if it is a subset of $R(X,\boldsymbol{Y})$ that satisfies the following for every X-value x that appears in R:

- If $\deg_R(\boldsymbol{Y}|X=x) \le \Delta$, then $\overline{R}$ contains all tuples of R with X-value x.
- If $\deg_R(\boldsymbol{Y}|X=x) > \Delta$, then $\overline{R}$ contains exactly Δ tuples of R with X-value x.

An (X,Δ)-restriction of R is not necessarily unique, as there can be several subsets of Δ tuples of R with X-value x.

Proposition 3.13 (Propagation of restrictions through RPQs). *Let $G = (V,E,\Sigma)$ be an input edge-labeled graph of size N and Δ a natural number. Consider the query*

$$Q(X,\boldsymbol{Z}) = \overset{\leftrightsquigarrow}{R}(X,Y) \wedge S(Y,\boldsymbol{Z}),$$

where $\overset{\leftrightsquigarrow}{R}(X,Y)$ is an RPQ over the alphabet Σ, and $S(Y,\boldsymbol{Z})$ is a relation over variables $\{Y\} \cup \boldsymbol{Z}$. Then, given a (Y,Δ)-restriction $\overline{S}(Y,\boldsymbol{Z})$ of $S(Y,\boldsymbol{Z})$, we can compute an (X,Δ)-restriction $\overline{Q}(X,\boldsymbol{Z})$ of $Q(X,\boldsymbol{Z})$ in time $O(N\cdot\Delta)$. [13]

Prop. 3.13 implies the following as a special case.

Proposition 3.14 ((X,Δ)-restriction of a single RPQ). *For any RPQ $\overset{\leftrightsquigarrow}{R}(X,Y)$ and any natural number Δ, we can compute an (X,Δ)-restriction $\bar{R}(X,Y)$ of $\overset{\leftrightsquigarrow}{R}(X,Y)$ in time $O(N\cdot\Delta)$.*

Prop. 3.14 follows from Prop. 3.13 because we can always construct an identity relation $S(Y,Z) \overset{\text{def}}{=} \{(v,v) \mid v \in V\}$ and apply Prop. 3.13 on the query $Q(X,Z) = \overset{\leftrightsquigarrow}{R}(X,Y) \wedge S(Y,Z)$.

Proposition 3.15 (Composition of (X,Δ)-restrictions). *Let $R_1(X,\boldsymbol{Z}_1)$ and $R_2(X,\boldsymbol{Z}_2)$ be two relations over variables $\{X\} \cup \boldsymbol{Z}_1$ and $\{X\} \cup \boldsymbol{Z}_2$, respectively, where $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ are* disjoint*. Consider the following query:*

$$Q(X,\boldsymbol{Z}_1,\boldsymbol{Z}_2) = R_1(X,\boldsymbol{Z}_1) \wedge R_2(X,\boldsymbol{Z}_2) \tag{18}$$

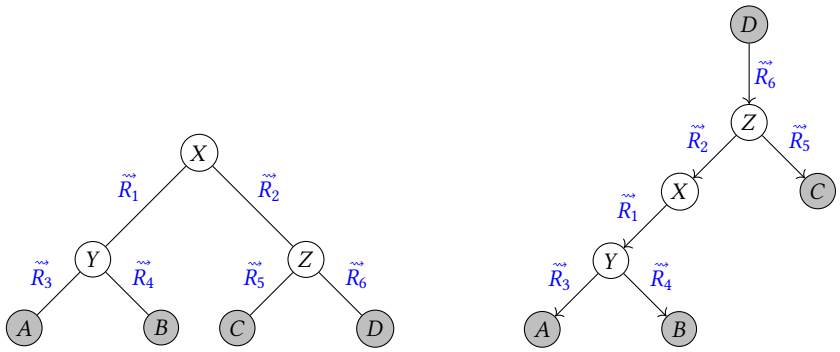
[13] The size of $\overline{S}$ could be much larger than N, yet, the runtime still holds. However, we also need to assume that $\overline{S}(Y,\boldsymbol{Z})$ is indexed by a *dictionary* that returns for a given X-value, the corresponding list of up to Δ $\boldsymbol{Z}$-tuples.

Given (X, Δ)-restrictions $\overline{R}_1(X, Z_1)$ of $R_1(X, Z_1)$ and $\overline{R}_2(X, Z_2)$ of $R_2(X, Z_2)$, where Δ is some natural number, an (X, Δ)-restriction $\overline{Q}(X, Z_1, Z_2)$ of $Q(X, Z_1, Z_2)$ can be computed in time $O(|Dom_X| \cdot \Delta)$, where Dom_X is the set of all X-values appearing in R_1 and R_2.

We next demonstrate the idea of the proof of Lemma 3.5 using the following example. We state the general proof in Appendix B.4.

Example 3.16 (for Lemma 3.5). Consider the following free-leaf CRPQ Q depicted in Fig. 2a:

$$Q(A, B, C, D) = \overset{\rightsquigarrow}{R}_1(X, Y) \wedge \overset{\rightsquigarrow}{R}_2(X, Z) \wedge \overset{\rightsquigarrow}{R}_3(Y, A) \wedge \overset{\rightsquigarrow}{R}_4(Y, B) \wedge \overset{\rightsquigarrow}{R}_5(Z, C) \wedge \overset{\rightsquigarrow}{R}_6(Z, D) \tag{19}$$



(a) The original query Q from Eq. (19). (b) The query Q rotated to have D as the root.

Fig. 2. The free-leaf acyclic CRPQ Q from Example 3.16. Free variables are shaded.

Pick an arbitrary free variable, say D, and re-orient the query tree so that D becomes the root, as shown in Fig. 2b. Note that when we re-orient an RPQ, e.g., $\overset{\rightsquigarrow}{R}_6(Z, D)$, so that it is from D to Z, the result is still an RPQ (Prop. 2.13). To reduce clutter, we abuse notation and refer to the re-oriented RPQ $\overset{\rightsquigarrow}{R}_6(Z, D)$ as $\overset{\rightsquigarrow}{R}_6(D, Z)$.

Using a bottom-up traversal of the query tree from the leaves to the root D, we can rewrite the query Q as a sequence of subqueries as follows:

$$\begin{aligned}
P_1(Y, A) &\overset{\text{def}}{=} \overset{\rightsquigarrow}{R}_3(Y, A) && \text{(Prop. 3.14)}\\
P_2(Y, B) &\overset{\text{def}}{=} \overset{\rightsquigarrow}{R}_4(Y, B) && \text{(Prop. 3.14)}\\
P_3(Y, A, B) &\overset{\text{def}}{=} P_1(Y, A) \wedge P_2(Y, B) && \text{(Prop. 3.15)}\\
P_4(X, A, B) &\overset{\text{def}}{=} \overset{\rightsquigarrow}{R}_1(X, Y) \wedge P_3(Y, A, B) && \text{(Prop. 3.13)}\\
P_5(Z, A, B) &\overset{\text{def}}{=} \overset{\rightsquigarrow}{R}_2(Z, X) \wedge P_4(X, A, B) && \text{(Prop. 3.13)}\\
P_6(Z, C) &\overset{\text{def}}{=} \overset{\rightsquigarrow}{R}_5(Z, C) && \text{(Prop. 3.14)}\\
P_7(Z, A, B, C) &\overset{\text{def}}{=} P_5(Z, A, B) \wedge P_6(Z, C) && \text{(Prop. 3.15)}\\
Q(D, A, B, C) &\overset{\text{def}}{=} \overset{\rightsquigarrow}{R}_6(D, Z) \wedge P_7(Z, A, B, C) && \text{(Prop. 3.13)}
\end{aligned}$$

We assume for now that we already know the output size OUT of Q upfront. We will show at the end of the example how to get rid of this assumption. Let $\Delta \overset{\text{def}}{=} \text{OUT}^{3/4}$. (In general, $\Delta \overset{\text{def}}{=}$

$\mathsf{OUT}^{1-1/|\mathsf{free}(Q)|}$.) Moreover, let $\Delta' \stackrel{\text{def}}{=} \Delta + 1$. For each query $P_i(X, \boldsymbol{Y})$, where X is the first variable and $\boldsymbol{Y}$ is the set of remaining variables, we compute an (X, Δ')-restriction $\overline{P}_i(X, \boldsymbol{Y})$ of $P_i(X, \boldsymbol{Y})$ using one of Props. 3.13, 3.14, or 3.15, in time $O(N \cdot \Delta')$. At the very end, we compute a (D, Δ')-restriction $\overline{Q}(D, A, B, C)$ of $Q(D, A, B, C)$. A vertex d is called *light* if $\deg_{\overline{Q}}(A, B, C|D = d) \leq \Delta$, and *heavy* otherwise (i.e., if $\deg_{\overline{Q}}(A, B, C|D = d) = \Delta' \stackrel{\text{def}}{=} \Delta + 1$; recall that $\deg_{\overline{Q}}(A, B, C|D = d)$ cannot exceed Δ' by definition of (D, Δ')-restriction). The relation $\overline{Q}$ already contains all output tuples (d, a, b, c) where d is light. Let H_D be the set of all heavy vertices d. Our next target is to report output tuples (d, a, b, c) where $d \in H_D$. Note that because each heavy vertex d in H_D has at least Δ associated tuples (a, b, c) where (d, a, b, c) is in the output, the number of heavy vertices, $|H_D|$, is at most $\mathsf{OUT}/\Delta = \mathsf{OUT}^{1/4}$. (In general, we get $|H_D| \leq \mathsf{OUT}^{1/|\mathsf{free}(Q)|}$.) Using Prop. 2.14, we can assume that the query $\overset{\leftrightsquigarrow}{R}_6(D, Z) \wedge H_D(D)$ is equivalent to an RPQ $\overset{\leftrightsquigarrow}{R}'_6(D, Z)$. Before we continue, we replace $\overset{\leftrightsquigarrow}{R}_6(D, Z)$ with $\overset{\leftrightsquigarrow}{R}'_6(D, Z)$, and from now on, we can assume that in the remaining query, the number of different D-values is at most $\mathsf{OUT}^{1/4}$.

Now, we pick another free variable, say C, and repeat the same process. Just like before, at the end of the bottom-up traversal, we will be able to report all output tuples (c, a, b, d) where c is light, and we will be left with a set of heavy vertices H_C of size at most $\mathsf{OUT}^{1/4}$. Our next target is to report output tuples (c, a, b, d) where $c \in H_C$ (and $d \in H_D$). In the remaining query, we can assume that the number of different C-values (and D-values) is at most $\mathsf{OUT}^{1/4}$.

Now, we pick another free variable, say B, and repeat the same process. After we report all output tuples (b, a, c, d) where b is light, we can assume that in the remaining query, the number of different B-values (as well as C-values and D-values) is at most $\mathsf{OUT}^{1/4}$.

Finally, we pick the last free variable, A. But now, since the number of different B-, C-, and D-values is at most $\mathsf{OUT}^{1/4}$, the total number of different tuples (b, c, d) cannot exceed $\Delta = \mathsf{OUT}^{3/4}$. Hence, in all remaining output tuples (a, b, c, d), the A-value a must be light, and we can report all of them at the end of the bottom-up traversal.

Guessing the output size OUT. We are left to explain how to guess the output size OUT if it is not given upfront. To that end, we start with an initial guess $\mathsf{OUT} = 1$ and keep doubling our guess every time we discover that our guess was below the actual output size. In particular, for each guess of OUT, we run the above algorithm. Consider the last bottom-up traversal above where the free variable A is the root. If it is indeed the case that all remaining A-values are light, then we already reported the whole output of Q and we are done. If not, then our guess of OUT was below the actual output size, and we need to double it and repeat the process. The total time spent is $O(\sum_{i=1,\ldots,\lceil \log \mathsf{OUT} \rceil} N \cdot 2^{3i/4})$. This is a geometric series that is dominated by its last term, which is $O(N \cdot \mathsf{OUT}^{3/4})$.

4 Related Work

The traditional approach for evaluating RPQs is based on taking the product graph (Definition 2.5) of the input graph with the NFA corresponding to the regular expression and then running a breadth-first search starting from each vertex in the product graph. In data complexity, this approach takes time $O(|V| \cdot |E|)$ [10, 26, 28], which is $O(N^2)$, where $N \stackrel{\text{def}}{=} |V| + |E|$. Moreover, no combinatorial algorithm can achieve any polynomial improvement[14] over this bound unless the *combinatorial Boolean matrix multiplication conjecture* fails [15]. A prior work introduced an output sensitive algorithm for RPQs whose complexity is $O(N + N \cdot \mathsf{OUT}^{1/2})$, where OUT is the output size [5].

[14]By that, we mean achieving a runtime of $O((|V| \cdot |E|)^{1-\epsilon})$ for any $\epsilon > 0$.

For CRPQs, prior work [16] proposed solving them by first materializing each RPQ separately, thus turning the CRPQ into a traditional CQ, and then evaluating this CQ using standard approaches. This approach is not output-sensitive, since the output of each individual RPQ could be much larger than the final output of the CRPQ. An output-sensitive alternative for *acyclic* CRPQs was proposed in [4]. It lifts the Yannakakis algorithm from CQs to CRPQs and also relies on the output-sensitive RPQ algorithm from [5]. See Appendix C for a detailed comparison with our work.

Recent work [20] studies the problem of *minimizing* a CRPQ, i.e., finding the smallest[15] equivalent CRPQ. This problem is not data-dependent, but it depends on the specific regular expressions used in the RPQ atoms of the CRPQ. In contrast, our results don't depend on the specific regular expressions. Nevertheless, minimizing a CRPQ can result in a smaller value of the free-connex fractional hypertree width, thus speeding up our algorithm.

The best known output-sensitive *combinatorial* algorithm for any acyclic CQ was recently proposed in [18, 24]. However, certain CQs are known to enjoy faster runtimes using non-combinatorial algorithms that rely on fast matrix multiplication. For example, Boolean matrix multiplication (which corresponds to $Q_{k\text{-path}}$ for $k = 2$) can be solved in time $O(N^{2/3} \cdot \mathsf{OUT}^{2/3} + N \cdot \mathsf{OUT}^{1/3})$ *assuming* $\omega = 2$ [9], where ω is the matrix multiplication exponent [36]. Recent work improved this bound to $O(N \cdot \mathsf{OUT}^{1/3} + \mathsf{OUT})$ [1]. The triangle query is another example on the cyclic side [12]. See [17] and [3] for further discussion on the use of fast matrix multiplication for CQ evaluation.

5 Conclusion and Future Directions

We presented an output sensitive algorithm for acyclic CRPQs that improves upon prior work. Moreover, despite CRPQs seemingly being more difficult than CQs due to their recursive nature, we show that our algorithm already matches the runtime of the best known output-sensitive combinatorial algorithm for the corresponding CQs. Hence, any further improvement on our algorithm can only happen modulo an improvement in the state-of-the-art on CQ evaluation.

Our result opens up several interesting directions, some of which are immediate extensions of our work, while others are more ambitious. We list some of them below:

Hybrid CQ/CRPQ queries. In practice, we might encounter conjunctive queries that are *hybrid* between CRPQs and CQs. In such queries, some atoms are RPQs $\overset{\leftrightsquigarrow}{R}(X, Y)$ while others are relational atoms $R(\boldsymbol{X})$ of arbitrary arities (not necessarily 2). Our results seem to *immediately* extend to such queries. To that end, we extend the definition of free-connex fractional hypertree width to those queries in the natural way. Since hybrid queries generalize CQs, the generalized algorithm would also subsume the one from [18, 24].

Aggregate RPQs and CRPQs. We can extend the semantics of RPQs and CRPQs to *weighted* edge-labeled graphs in a natural way as follows: Consider an input edge-labeled graph where each edge has a weight, e.g., a non-negative real number. Given an RPQ $\overset{\leftrightsquigarrow}{R}(X, Y)$, can we list all pairs of vertices x and y such that there is a path from x to y whose label matches $\overset{\leftrightsquigarrow}{R}$, *and* also list the *minimum total weight* of such a path from x to y (i.e., the shortest path length)? To make the problem more general, we can choose any semiring $\boldsymbol{K}$ and consider a $\boldsymbol{K}$*-annotated* edge-labeled graph, i.e., an edge-labeled graph where each edge is annotated with an element from $\boldsymbol{K}$. Given a path in such graph, we define its annotation as the product of the annotations of its edges (using the multiplication operation of $\boldsymbol{K}$). Then, the semantics of an RPQ $\overset{\leftrightsquigarrow}{R}(X, Y)$ over such a graph is as follows: For each pair of vertices x and y, compute a sum (using the addition operation of $\boldsymbol{K}$) of the annotations of all paths from x to y whose label is in the language defined by $\overset{\leftrightsquigarrow}{R}$. This semantics

[15]This typically means finding a CRPQ with the minimum number of atoms.

naturally extends to CRPQs as well, and also mirrors nicely the traditional semantics of aggregate CQs over $\boldsymbol{K}$-annotated databases [22]. In the CQ world, prior output-sensitive results [18, 24] work for aggregate CQs over $\boldsymbol{K}$-annotated databases out-of-the-box. But now we ask: can we extend our result to aggregate CRPQs over $\boldsymbol{K}$-annotated edge-labeled graphs? The answer doesn't seem to be as straightforward as in the CQ case, and it also seems to depend on the choice of the semiring $\boldsymbol{K}$:

- For the tropical semiring $(\min, +)$, this seems to be possible. The main change in our algorithm that needs to be done is as follows: Currently, Proposition 3.13 uses a breadth-first search; we need to replace it with Dijkstra's algorithm to compute shortest paths.
- For the $(+, \times)$ semiring, the problem can be a lot more challenging. In particular, it subsumes counting the number of paths in a graph, where these paths are of *unbounded* length. This is in contrast to the CQ world, where all paths have constant length in data complexity. It seems unlikely that we can still achieve the same runtime bound under this semiring.

We leave characterizing the general *semiring-dependent* complexity of aggregate CRPQs as an open problem for future work.

Using Fast Matrix Multiplication to speed up CRPQ evaluation. On the acyclic CQ side, certain queries admit non-combinatorial algorithms that utilize fast matrix multiplication to beat the runtimes from [18, 24]. Among those queries is the Boolean matrix multiplication query [1, 9], which corresponds to $Q_{k\text{-path}}$ for $k = 2$ (Eq. (4)). It would be interesting to see if similar techniques can be leveraged to improve the output-sensitive runtimes for acyclic CRPQs as well.

Cyclic CRPQs. Cyclic CRPQs seems to be far more challenging than their cyclic CQ counterparts. For example, consider the (Boolean) triangle CRPQ: $Q() = \overset{\rightsquigarrow}{R_1}(X, Y) \wedge \overset{\rightsquigarrow}{R_2}(Y, Z) \wedge \overset{\rightsquigarrow}{R_3}(Z, X)$. While it might be tempting at first to try to mirror the success of Worst-case Optimal Join Algorithms for cyclic CQs [29–31, 35], this seems to be a lot more challenging for CRPQs. This is because the k-expansion (Definition 2.10) of the triangle is a k'-cycle where $k' \overset{\text{def}}{=} 3k$. In turn, as k' goes to infinity, the best known complexity for the k'-cycle CQ approaches $O(N^2)$ [7, 8], which is not a very interesting complexity for the triangle CRPQ. We leave characterizing the complexity of cyclic CRPQs, including proving lower bounds, as an open question.

Acknowledgments

Suciu's work was partially supported by NSF IIS 2314527, NSF SHF 2312195, NSF III 2507117, and a Microsoft professorship. Olteanu's work is partially supported by SNSF 200021-231956.

References

[1] Amir Abboud, Karl Bringmann, Nick Fischer, and Marvin Künnemann. 2024. The Time Complexity of Fully Sparse Matrix Multiplication. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 4670–4703.

[2] Mahmoud Abo Khamis, Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2020. Functional Aggregate Queries with Additive Inequalities. *ACM Trans. Database Syst.* 45, 4 (2020), 17:1–17:41. doi:10.1145/3426865

[3] Mahmoud Abo Khamis, Xiao Hu, and Dan Suciu. 2025. Fast Matrix Multiplication meets the Submodular Width. *Proc. ACM Manag. Data* 3, 2, Article 98 (June 2025), 26 pages. doi:10.1145/3725235

[4] Mahmoud Abo Khamis, Alexandru-Mihai Hurjui, Ahmet Kara, Dan Olteanu, Dan Suciu, and Zilu Tian. 2025. Output-Sensitive Evaluation of Acyclic Conjunctive Regular Path Queries. *arXiv e-prints*, Article arXiv:2509.20204 (Sept. 2025), arXiv:2509.20204 pages. arXiv:2509.20204 [cs.DB] doi:10.48550/arXiv.2509.20204

[5] Mahmoud Abo Khamis, Ahmet Kara, Dan Olteanu, and Dan Suciu. 2025. Output-Sensitive Evaluation of Regular Path Queries. 3, 2, Article 105 (June 2025), 20 pages. doi:10.1145/3725242

[6] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. 2016. FAQ: Questions Asked Frequently. In *PODS*. 13–28. doi:10.1145/2902251.2902280

[7] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2025. PANDA: Query Evaluation in Submodular Width. *TheoretiCS* Volume 4, Article 12 (Apr 2025). doi:10.46298/theoretics.25.12

[8] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-Coding. *J. ACM* 42, 4 (1995), 844–856. doi:10.1145/210332.210337

[9] Rasmus Resen Amossen and Rasmus Pagh. 2009. Faster Join-Projects and Sparse Matrix Multiplications. In *ICDT*. 121–126. doi:10.1145/1514894.1514909

[10] Pablo Barceló Baeza. 2013. Querying Graph Databases. In *PODS*. 175–188. doi:10.1145/2463664.2465216

[11] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On Acyclic Conjunctive Queries and Constant Delay Enumeration. In *CSL*. 208–222. doi:10.1007/978-3-540-74915-8_18

[12] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. 2014. Listing Triangles. In *ICALP*. 223–234. doi:10.1007/978-3-662-43948-7_19

[13] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An analytical study of large SPARQL query logs. *Proc. VLDB Endow.* 11, 2 (Oct. 2017), 149–161. doi:10.14778/3149193.3149196

[14] Angela Bonifati, Wim Martens, and Thomas Timm. 2019. Navigating the Maze of Wikidata Query Logs. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 127–138. doi:10.1145/3308558.3313472

[15] Katrin Casel and Markus L. Schmid. 2023. Fine-Grained Complexity of Regular Path Queries. *LMCS* 19, 4 (2023). doi:10.46298/LMCS-19(4:15)2023

[16] Tamara Cucumides, Juan L. Reutter, and Domagoj Vrgoc. 2023. Size Bounds and Algorithms for Conjunctive Regular Path Queries. In *ICDT*. 13:1–13:17. doi:10.4230/LIPICS.ICDT.2023.13

[17] Shaleen Deep, Xiao Hu, and Paraschos Koutris. 2020. Fast Join Project Query Evaluation using Matrix Multiplication. In *SIGMOD*. 1213–1223. doi:10.1145/3318464.3380607

[18] Shaleen Deep, Hangdong Zhao, Austen Z. Fan, and Paraschos Koutris. 2024. Output-sensitive Conjunctive Query Evaluation. 2, 5, Article 220 (Nov. 2024), 24 pages. doi:10.1145/3695838

[19] Diego Figueira, Adwait Godbole, S. Krishna, Wim Martens, Matthias Niewerth, and Tina Trautner. 2020. Containment of Simple Conjunctive Regular Path Queries. In *KR*. 371–380. doi:10.24963/KR.2020/38

[20] Diego Figueira, Rémi Morvan, and Miguel Romero. 2025. Minimizing Conjunctive Regular Path Queries. *Proc. ACM Manag. Data* 3, 2, Article 100 (June 2025), 25 pages. doi:10.1145/3725237

[21] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In *SIGMOD*. 1433–1445. doi:10.1145/3183713.3190657

[22] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Beijing, China) *(PODS '07)*. Association for Computing Machinery, New York, NY, USA, 31–40. doi:10.1145/1265530.1265535

[23] Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 Query Language. W3C Recommendation. http://www.w3.org/TR/sparql11-query/.

[24] Xiao Hu. 2025. Output-Optimal Algorithms for Join-Aggregate Queries. *Proc. ACM Manag. Data* 3, 2, Article 104 (June 2025), 27 pages. doi:10.1145/3725241

[25] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. 2018. Getting the Most Out of Wikidata: Semantic Technology Usage in Wikipedia's Knowledge Graph. In *The Semantic Web – ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II* (Monterey, CA, USA). Springer-Verlag, Berlin, Heidelberg, 376–394. doi:10.1007/978-3-030-00668-6_23

[26] Wim Martens and Tina Trautner. 2018. Evaluation and Enumeration Problems for Regular Path Queries. In *ICDT*. 19:1–19:21. doi:10.4230/LIPICS.ICDT.2018.19

[27] Wim Martens and Tina Trautner. 2019. Bridging Theory and Practice with Query Log Analysis. *SIGMOD Rec.* 48, 1 (2019), 6–13. doi:10.1145/3371316.3371319

[28] Alberto O. Mendelzon and Peter T. Wood. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM J. Comput.* 24, 6 (1995), 1235–1258. doi:10.1137/S009753979122370X

[29] Hung Q. Ngo. 2018. Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. In *PODS*. 111–124. doi:10.1145/3196959.3196990

[30] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case Optimal Join Algorithms. *J. ACM* 65, 3 (2018), 16:1–16:40. doi:10.1145/2213556.2213565

[31] Hung Q Ngo, Christopher Ré, and Atri Rudra. 2014. Skew Strikes Back: New Developments in the Theory of Join Algorithms. *SIGMOD Rec.* 42, 4 (feb 2014), 5–16. doi:10.1145/2590989.2590991

[32] Dan Olteanu and Jakub Závodný. 2015. Size Bounds for Factorised Representations of Query Results. *ACM Trans. Database Syst.* 40, 1 (2015), 2:1–2:44. doi:10.1145/2656335

[33] Luc Segoufin. 2013. Enumerating with constant delay the answers to a query. In *Proceedings of the 16th International Conference on Database Theory* (Genoa, Italy) *(ICDT '13)*. Association for Computing Machinery, New York, NY, USA,

10–20. doi:10.1145/2448496.2448498
[34] Volker Strassen. 1969. Gaussian elimination is not optimal. *Numer. Math.* 13 (1969), 354–356. https://api.semanticscholar.org/CorpusID:121656251
[35] Todd L. Veldhuizen. 2014. Triejoin: A Simple, Worst-Case Optimal Join Algorithm. In *ICDT*. 96–106. doi:10.5441/002/ICDT.2014.13
[36] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. [n. d.]. *New Bounds for Matrix Multiplication: from Alpha to Omega.* 3792–3835. arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611977912.134 doi:10.1137/1.9781611977912.134
[37] Mihalis Yannakakis. 1981. Algorithms for Acyclic Database Schemes. In *VLDB*. 82–94.

A Missing details in Section 2

A.1 Proof of Proposition 2.12

Proposition 2.12. *(The CRPQ $Q(X) = \overset{\rightsquigarrow}{R}(X,Y)$ is evaluatable in linear time.) Given an edge-labeled graph of size N, the CRPQ $Q(X) = \overset{\rightsquigarrow}{R}(X,Y)$ can be evaluated in $O(N)$ time.*

Proof. Assume that the CRPQ Q is defined over an alphabet Σ, and let $G = (V, E, \Sigma)$ be an input edge-labeled graph for Q. Let $M_{\overset{\rightsquigarrow}{R}} = (V_{\overset{\rightsquigarrow}{R}}, E_{\overset{\rightsquigarrow}{R}}, \Sigma)$ be an edge-labeled graph representing the nondeterministic finite automaton (NFA) for the regular expression $\overset{\rightsquigarrow}{R}$. Let $G' = (V', E')$ be the product graph of G and $M_{\overset{\rightsquigarrow}{R}}$ (Definition 2.5). Recall that $V' = V \times V_{\overset{\rightsquigarrow}{R}}$. Let $T \subseteq V'$ be the set of all vertices in V' of the form (v, q_f) where $v \in V$ and q_f is an accepting state of $M_{\overset{\rightsquigarrow}{R}}$. Similarly, let S be the set of all vertices in V' of the form (u, q_0) where $u \in V$ and q_0 is the initial state of $M_{\overset{\rightsquigarrow}{R}}$. To compute the answer of Q on G, we do a *backward* traversal of G' starting from all vertices in T, until we compute the subset $S' \subseteq S$ of vertices in S reachable from T. The answer of Q on G is then obtained by projecting each vertex in S' onto its first component.

Since the size of Q, and hence of $\overset{\rightsquigarrow}{R}$, is constant, the product graph G' can be constructed in $O(|V| + |E|) = O(N)$ time. Traversal takes $O(N)$ time as well. □

A.2 Proof of Proposition 2.14

Proposition 2.14. *(The query $Q(X,Y) = \overset{\rightsquigarrow}{R}(X,Y) \wedge S(Y)$ is equivalent to an RPQ.) Let $\overset{\rightsquigarrow}{R}(X,Y)$ be an RPQ over an edge-labeled graph $G = (V, E, \Sigma)$, and $S \subseteq V$ be a set of vertices viewed as a unary relation. The answer of the query $Q(X,Y) = \overset{\rightsquigarrow}{R}(X,Y) \wedge S(Y)$ on G is the same as the answer of an RPQ $\overset{\rightsquigarrow}{R'}(X,Y)$ on an edge-labeled graph $G' = (V, E', \Sigma')$ that can be constructed from G in linear time.*

Proof. Let σ' be a fresh symbol not in Σ, and $\Sigma' \stackrel{\text{def}}{=} \Sigma \cup \{\sigma'\}$. We construct $G' = (V, E', \Sigma')$ by extending the edge set E with self-loops at every vertex $s \in S$ with label σ'. Now, let the regular expression $\overset{\rightsquigarrow}{R'}$ be the concatenation of the regular expression $\overset{\rightsquigarrow}{R}$ and the symbol σ'. Then, answering the RPQ $\overset{\rightsquigarrow}{R'}(X,Y)$ over G' is equivalent to answering the original query $Q(X,Y)$ over G. □

A.3 Proof of Proposition 2.18

Proposition 2.18 (fn-fhtw of k-expansion). *Given an acyclic CRPQ Q and a natural number $k \geq 2$, let Q'_k be the k-expansion of Q (Definition 2.10). The following holds:*

$$\textit{fn-fhtw}(Q'_k) \quad = \quad \begin{cases} 1 & \text{if } Q \text{ is trivial,} \\ \max(\textit{fn-fhtw}(Q), 2) & \text{if } Q \text{ is non-trivial.} \end{cases} \tag{10}$$

Proof. Let Q be an acyclic CRPQ and $k \geq 2$ be any natural number. Let Q' be the k-expansion of Q (Definition 2.10). If Q is trivial, then it is free-connex, hence Q' is also free-connex. By Proposition 2.16, $\text{fn-fhtw}(Q') = 1$.

Now, we consider the case where Q is non-trivial. Let $Q_1, \ldots, Q_m$ be the bound-connected components of Q (Definition 3.7). Since Q is acyclic, each Q_i is also acyclic. Moreover, let $Q'_1, \ldots, Q'_m$ be the k-expansions of $Q_1, \ldots, Q_m$, respectively. It is straightforward to verify that $Q'_1, \ldots, Q'_m$ are the bound-connected components of Q'.

Claim A.1. Let Q be a acyclic bound-connected CRPQ (or acyclic bound-connected CQ with only binary atoms). Let Q' be its k-expansion for some $k \geq 2$. We call Q *single-edge* if it consists of a single atom $\overset{\leftrightsquigarrow}{R}(X, Y)$ where both X and Y are free variables. It must hold that

$$\rho^*_Q(\mathsf{free}(Q)) = \begin{cases} 1 & \text{if } Q \text{ is single-edge} \\ |\mathsf{free}(Q)| & \text{otherwise} \end{cases} \tag{20}$$

$$\rho^*_{Q'}(\mathsf{free}(Q')) = |\mathsf{free}(Q)| \tag{21}$$

Assume for now that the above claim holds.

- If no Q_i is a single-edge, then $\rho^*_{Q_i}(\mathsf{free}(Q_i)) = \rho^*_{Q'_i}(\mathsf{free}(Q'_i))$ for every $i \in [m]$. Lemma 3.8 implies that $\mathsf{fn\text{-}fhtw}(Q) = \mathsf{fn\text{-}fhtw}(Q')$. Moreover, since Q is non-trivial, one Q_i must contain at least two free variables. Hence, $\mathsf{fn\text{-}fhtw}(Q) \geq 2$ and $\mathsf{fn\text{-}fhtw}(Q') = \max(\mathsf{fn\text{-}fhtw}(Q), 2)$ holds as well.
- If some Q_i is a single-edge, then $\rho^*_{Q_i}(\mathsf{free}(Q_i)) = 1$ and $\rho^*_{Q'_i}(\mathsf{free}(Q'_i)) = 2$. Therefore, by Lemma 3.8, $\mathsf{fn\text{-}fhtw}(Q') = \max(\mathsf{fn\text{-}fhtw}(Q), 2)$.

Proof of Claim A.1. The above claim is proved similarly to Proposition 3.9. In particular, because Q is acyclic, bound-connected, where all atoms are binary, Q must be a tree where free variables can occur only at the leaves. The only way for a single edge to cover two free variables is when two leaves are directly connected by an edge, which can only happen when the query is a single edge. In all other cases, each free variable must be covered by a distinct edge. This proves Eq. (20).

To prove Eq. (21), note that Q' must also be acyclic, bound-connected, where all atoms are binary, and $\mathsf{free}(Q') = \mathsf{free}(Q)$. Moreover, Q' cannot be single-edge, even when Q is single-edge. (Recall that $k \geq 2$.) □

□

B Missing details in Section 3

B.1 Proof of Lemma 3.8

Lemma 3.8 ([24]). *Let Q be an acyclic CRPQ (or acyclic CQ) and $Q_1, \ldots, Q_k$ its bound-connected components.*[16] *Then, we have*

$$\mathit{fn\text{-}fhtw}(Q) = \max(\max_{i \in [k]} \rho^*_{Q_i}(\mathit{free}(Q_i)), 1) \tag{16}$$

Below, we only prove the $\geq$ direction in Eq. (16) because our main result, Theorem 3.1, only uses this direction. The $\leq$ direction is exclusively used in the proof of Proposition 2.18, and we refer the reader to [24] (Lemmas 3.3 and 3.4) for a proof of that direction.

Proof of $\geq$ direction in Lemma 3.8. We start with some definitions. A *(multi-)hypergraph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a finite set of vertices $\mathcal{V}$ and a (multi-)set of edges $\mathcal{E}$ such that each edge is a subset of $\mathcal{V}$ (of arbitrary size). Given a hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a vertex $v \in \mathcal{V}$, we

[16] The bound-connected components of an acyclic CRPQ are also acyclic.

define the sets $\mathcal{E}_v = \{e \in \mathcal{E} \mid v \in e\}$ and $\mathcal{V}_v = \bigcup_{e \in \mathcal{E}_v} e$. The hypergraph obtained by the elimination of v in $\mathcal{G}$ is defined as $\text{eliminate}(\mathcal{G}, v) = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}' = \mathcal{V} \setminus \{v\}$ and $\mathcal{E}' = (\mathcal{E} \setminus \mathcal{E}_v) \cup \{\mathcal{V}_v \setminus \{v\}\}$.

Next, we recall the definition of variable elimination orders from prior work [2, 6].

Definition B.1 (Variable Elimination Orders)**.** A *variable elimination order* for a CRPQ Q is an ordering $\sigma = (X_1, \ldots, X_n)$ of its variables. The *hypergraph sequence induced* by σ is of the form $(\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_n)$, where $\mathcal{G}_0$ is the query graph of Q and $\mathcal{G}_i = \text{eliminate}(\mathcal{G}_{i-1}, X_i)$ for $i \in [n]$.[17] The *variable set sequence* induced by σ is $(\mathcal{V}_{X_1}, \ldots, \mathcal{V}_{X_n})$. A variable elimination order is called *bound-leading* if no free variable appears before any bound one.

Prior work shows:

Lemma B.2 ([2, 6]). *Let Q be a CRPQ. For any free-connex tree decomposition (T, χ) for Q, there is a bound-leading elimination order for Q such that every set in the variable set sequence induced by σ is contained in a bag $\chi(t)$ for some $t \in \text{nodes}(T)$.*

To prove Lemma 3.8, we use the following two auxiliary lemmas.

Lemma B.3. *Let Q be an acyclic CRPQ with bound-connected components $Q_1, \ldots, Q_k$ and let $(X_1, \ldots, X_n)$ be a bound-leading variable elimination order for Q with induced variable set sequence $(\mathcal{V}_{X_1}, \ldots, \mathcal{V}_{X_n})$. For any $i \in [k]$, there is a $j \in [n]$ such that $\text{free}(Q_i) \subseteq \mathcal{V}_{X_j}$.*

Proof. Consider an acyclic CRPQ Q. Let $(X_1, \ldots, X_n)$ be a bound-leading variable elimination order for Q with induced variable set sequence $(\mathcal{V}_{X_1}, \ldots, \mathcal{V}_{X_n})$ and induced hypergraph sequence $(\mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_n)$. Let $(X_1, \ldots, X_m)$ be the bound variables in Q. The main observation is that for any bound variable X_j, it holds: if $\mathcal{V}_{X_j}$ contains two free variables, then they must be contained in the same bound-connected component of Q. Consider the hypergraph $\mathcal{G}_m$, which we obtain after eliminating all bound variables. Each hyperedge in $\mathcal{G}_m$ is a set in the induced variable set sequence and must consist of the free variables of a bound-connected component of Q. Hence the free variables of each bound-connected component of Q must be contained in a set of the induced variable set sequence. □

Lemma B.4. *Let Q be an acyclic CRPQ and Q' a bound-connected component of Q. It holds $\rho^*_Q(\text{free}(Q')) = \rho^*_{Q'}(\text{free}(Q'))$.*

Proof. Let Q be an acyclic CRPQ and Q' a bound-connected component of Q. Since $\text{atoms}(Q) \supseteq \text{atoms}(Q')$, it holds $\rho^*_Q(\text{free}(Q')) \leq \rho^*_{Q'}(\text{free}(Q'))$. Hence, it suffices to show that $\rho^*_Q(\text{free}(Q')) \geq \rho^*_{Q'}(\text{free}(Q'))$. For the sake of contradiction, assume that $\rho^*_Q(\text{free}(Q')) < \rho^*_{Q'}(\text{free}(Q'))$. This means that Q contains an atom $\overset{\leftrightarrow}{R}(X, Y)$ such that $\overset{\leftrightarrow}{R}(X, Y)$ is not contained in Q' and X and Y are two distinct free variables in Q'. Since each pair of variables in Q' is connected, this implies that Q is cyclic, which is a contradiction to our initial assumption. □

We are ready to prove Lemma 3.8. Consider an acyclic CRPQ Q with bound-connected components $Q_1, \ldots, Q_k$ and query graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Consider a free-connex tree decomposition (T, χ) for Q that is *optimal*, i.e., that satisfies $\max_{t \in \text{nodes}(T)} \rho^*_Q(\chi(t)) = \text{fn-fhtw}(Q)$. By Lemma B.2, there exists a bound-leading elimination order $(X_1, \ldots, X_n)$ for Q with induced variable set sequence $(\mathcal{V}_{X_1}, \ldots, \mathcal{V}_{X_n})$ such that for every $j \in [n]$, there is $t \in \text{nodes}(T)$ with $\mathcal{V}_{X_j} \subseteq \chi(t)$. This implies that $\text{fn-fhtw}(Q) \geq \max_{j \in [n]} \rho^*_Q(\mathcal{V}_{X_j})$. By Lemma B.3, it holds that for every $i \in [k]$, there is a $j \in [n]$ such that $\text{free}(Q_i) \subseteq \mathcal{V}_{X_j}$. This implies that $\text{fn-fhtw}(Q) \geq \max_{i \in [k]} \rho^*_Q(\text{free}(Q_i))$. By Lemma B.4, it

[17] In the original formulation, the elimination proceeds from the last variable to the first in the elimination order, that is, from right to left [6]. For notational convenience, we instead traverse the elimination order from left to right.

holds $\rho^*_Q(\text{free}(Q_i)) = \rho^*_{Q_i}(\text{free}(Q_i))$ for $i \in [k]$. This means $\text{fn-fhtw}(Q) \geq \max_{i\in[k]} \rho^*_{Q_i}(\text{free}(Q_i))$. Moreover, $\text{fn-fhtw}(Q) \geq 1$ by definition. This proves the $\geq$ direction in Eq. (16). □

B.2 Proof of Proposition 3.9

Proposition 3.9. *For any acyclic bound-connected CRPQ Q, $|free(Q)| \leq \max(\rho^*_Q(free(Q)), 2)$.*

Proof. Since Q is acyclic, bound-connected, where all atoms are binary, Q must be a tree where free variables can occur only at the leaves. In general, we need a distinct edge to cover each free leaf, i.e., $\rho^*_Q(\text{free}(Q)) = |\text{free}(Q)|$, *unless* there are two free leaves that are directly connected by an edge, which can only happen when the query consists of a single edge. In this case, we have $\rho^*_Q(\text{free}(Q)) = 1$ and $|\text{free}(Q)| = 2$. □

B.3 Proof of Proposition 3.13

Proposition 3.13 (Propagation of restrictions through RPQs). *Let $G = (V, E, \Sigma)$ be an input edge-labeled graph of size N and Δ a natural number. Consider the query*

$$Q(X, Z) = \overset{\rightsquigarrow}{R}(X, Y) \wedge S(Y, Z),$$

where $\overset{\rightsquigarrow}{R}(X, Y)$ is an RPQ over the alphabet Σ, and $S(Y, Z)$ is a relation over variables $\{Y\} \cup Z$. Then, given a (Y, Δ)-restriction $\overline{S}(Y, Z)$ of $S(Y, Z)$, we can compute an (X, Δ)-restriction $\overline{Q}(X, Z)$ of $Q(X, Z)$ in time $O(N \cdot \Delta)$. [18]

Proof. Let $M_{\overset{\rightsquigarrow}{R}} = (V_{\overset{\rightsquigarrow}{R}}, E_{\overset{\rightsquigarrow}{R}}, \Sigma)$ be an edge-labeled graph representing the NFA for the regular expression $\overset{\rightsquigarrow}{R}$. Let $G' = (V', E')$ be the product graph of G and $M_{\overset{\rightsquigarrow}{R}}$ (Definition 2.5). For each vertex $(v, q) \in (V' \overset{\text{def}}{=} V \times V_{\overset{\rightsquigarrow}{R}})$ in G', we compute a list, $\text{list}(v, q)$, of up to Δ distinct Z-tuples z such that there is a path in G' from (v, q) to some $(y, q_f) \in V'$ where q_f is an accepting state of $M_{\overset{\rightsquigarrow}{R}}$ and $(y, z) \in \overline{S}$. This computation can be done in time $O(N \cdot \Delta)$ as follows:

- For each vertex $(y, q_f) \in V'$ where q_f is an accepting state of $M_{\overset{\rightsquigarrow}{R}}$, we initialize $\text{list}(y, q_f)$ to be the set of all Z-tuples z such that $(y, z) \in \overline{S}$.
- Every time we add a Z-tuple z to $\text{list}(v, q)$ for some vertex $(v, q) \in V'$, we go through all incoming edges $((v_2, q_2), (v, q))$ and we add z to $\text{list}(v_2, q_2)$ *assuming* z is not already in this list and $|\text{list}(v_2, q_2)| < \Delta$. We repeat this process until no more tuples can be added to any list.

During the above process, each edge $((v_2, q_2), (v, q)) \in E'$ is traversed once every time a tuple z is added to $\text{list}(v, q)$, and there can only be at most Δ such tuples. Hence, the overall runtime is $O(N \cdot \Delta)$. Finally, we set $\overline{Q}(X, Z)$ as follows, where q_0 is the initial state of $M_{\overset{\rightsquigarrow}{R}}$:

$$\overline{Q}(X, Z) \quad \overset{\text{def}}{=} \quad \{(x, z) \mid x \in V \text{ and } z \in \text{list}(x, q_0)\}$$

□

B.4 Proof of Lemma 3.5

Lemma 3.5 (Solving free-leaf CRPQs). *Let Q be any free-leaf CRPQ, and G be an input edge-labeled graph of size N. Then, Q can be answered in time $O(N + N \cdot OUT^{1-\frac{1}{|free(Q)|}})$ in data complexity, where OUT is the output size.*

[18] The size of $\overline{S}$ could be much larger than N, yet, the runtime still holds. However, we also need to assume that $\overline{S}(Y, Z)$ is indexed by a *dictionary* that returns for a given X-value, the corresponding list of up to Δ Z-tuples.

Proof. Let Q be any free-leaf CRPQ with ℓ leaves, i.e., $\ell \stackrel{\text{def}}{=} |\mathsf{free}(Q)|$, and let G be an input edge-labeled graph of size N. We assume for now that the output size OUT is known upfront; we will later explain how to get rid of this assumption. Let $\Delta \stackrel{\text{def}}{=} \mathsf{OUT}^{1-1/\ell}$, and $\Delta' \stackrel{\text{def}}{=} \Delta + 1$. Pick any free variable $W \in \mathsf{free}(Q)$, and re-orient the query tree so that W becomes the root. For each variable $Y \in \mathsf{vars}(Q)$, let $\boldsymbol{F}_Y \subseteq \mathsf{free}(Q)$ be the set of free variables in the subtree rooted at Y excluding W (which can happen if Y is the root W). We do a bottom-up traversal of the query tree, from the leaves to the root, W, where for each variable Y, let X be its parent variable (assuming Y is not the root), and let $\overset{\rightsquigarrow}{R}(X, Y)$ be the RPQ associated with the edge (X, Y) in the query graph. For each variable Y, we define subqueries *without* actually computing them yet, as follows:

- If Y is a leaf, we define the following subquery: (Recall that Y must be a free variable in this case because Q is free-leaf.)

$$T_Y(X, Y) \quad \stackrel{\text{def}}{=} \quad \overset{\rightsquigarrow}{R}(X, Y) \qquad \text{(Prop. 3.14)}$$

- Otherwise, Let $Z_1, \ldots, Z_k$ be the children of Y in the query tree. We define the following two subqueries: (Note that in this case, $\boldsymbol{F}_{Z_1}, \ldots, \boldsymbol{F}_{Z_k}$ must be disjoint and their union must be $\boldsymbol{F}_Y$.)

$$S_Y(Y, \boldsymbol{F}_Y) \quad \stackrel{\text{def}}{=} \quad T_{Z_1}(Y, \boldsymbol{F}_{Z_1}) \wedge \cdots \wedge T_{Z_k}(Y, \boldsymbol{F}_{Z_k}) \qquad \text{(Prop. 3.15)}$$

$$T_Y(X, \boldsymbol{F}_Y) \quad \stackrel{\text{def}}{=} \quad \overset{\rightsquigarrow}{R}(X, Y) \wedge S_Y(Y, \boldsymbol{F}_Y) \qquad \text{(Prop. 3.13)}$$

By induction, it is straightforward to verify the following two claims:

Claim B.5. For each variable Y, $T_Y(X, \boldsymbol{F}_Y)$ is equivalent to a subquery of the CRPQ Q with free variables $\{X\} \cup \boldsymbol{F}_Y$ and with edges corresponding to the subtree rooted at Y in addition to the edge (X, Y).

Claim B.6. For each non-leaf variable Y, $S_Y(Y, \boldsymbol{F}_Y)$ is equivalent to a subquery of the CRPQ Q with free variables $\{Y\} \cup \boldsymbol{F}_Y$ and with edges corresponding to the subtree rooted at Y.

As a result, $S_W(W, \boldsymbol{F}_W)$ is equivalent to the original query Q. (Note that $\{W\} \cup \boldsymbol{F}_W = \mathsf{free}(Q)$.) By applying Prop. 3.14, we can compute an (X, Δ')-restriction $\overline{T}_Y(X, Y)$ of $T_Y(X, Y)$ for each leaf variable Y in time $O(N \cdot \Delta')$. Then, by repeatedly alternating Props. 3.15 and 3.13, we can compute a (Y, Δ')-restriction $\overline{S}_Y(Y, \boldsymbol{F}_Y)$ of $S_Y(Y, \boldsymbol{F}_Y)$ and an (X, Δ')-restriction $\overline{T}_Y(X, \boldsymbol{F}_Y)$ of $T_Y(X, \boldsymbol{F}_Y)$ for each non-leaf variable Y in time $O(N \cdot \Delta')$. At the very end, we compute a (W, Δ')-restriction $\overline{Q}(W, \boldsymbol{F}_W)$ of $Q(W, \boldsymbol{F}_W)$ in time $O(N \cdot \Delta')$. A W-value w is called *light* if $\deg_{\overline{Q}}(\boldsymbol{F}_W | W = w) \le \Delta$, and *heavy* otherwise (i.e., if $\deg_{\overline{Q}}(\boldsymbol{F}_W | W = w) = \Delta' \stackrel{\text{def}}{=} \Delta + 1$). The relation $\overline{Q}$ already contains all output tuples where the W-value is light. Let H_W be the set of all heavy W-values. The size of H_W is at most $\mathsf{OUT}/\Delta = \mathsf{OUT}^{1/\ell}$. Our next target is to report output tuples where the W-value is heavy. Using Prop. 2.14, we can "merge" the filter H_W into any RPQ that has W as an endpoint.

We repeat the above process for each free variable in $\mathsf{free}(Q)$. Every time we repeat the process, there will be one extra free variable W whose number of different values is at most $\mathsf{OUT}^{1/\ell}$. Finally, let Z be the *last* free variable we pick. At this point, every other free variable has at most $\mathsf{OUT}^{1/\ell}$ different values. Let $\boldsymbol{F}_Z \stackrel{\text{def}}{=} \mathsf{free}(Q) \setminus \{Z\}$. The total number of different $\boldsymbol{F}_Z$-tuples is at most $(\mathsf{OUT}^{1/\ell})^{\ell-1} = \Delta$. Hence, at this point, all remaining output tuples must have light Z-values, and we can report all of them during the last bottom-up traversal.

The above algorithm was under the assumption that the output size OUT is known upfront. If this is not the case, then we start by guessing $\mathsf{OUT} = 1$ and keep doubling our guess every time we discover that our guess was below the actual output size, similar to Example 3.16. The total

runtime would be $\sum_{i=1,\ldots,\lceil \log \mathsf{OUT} \rceil} N \cdot 2^{(\ell-1)i/\ell}$. This is a geometric series that is dominated by its last term, which is $O(N \cdot \mathsf{OUT}^{(\ell-1)/\ell})$. □

C Comparison with Existing Work

In this section, we compare the performance of our algorithm with the performance of two prior approaches for evaluating CRPQs. We use the following metrics in our complexity expressions: N is the size of the edge-labeled graph; OUT is the output size of the query; OUT_a is the maximal size of the (uncalibrated) output of any RPQ appearing in the query; fn-fhtw is the free-connex fractional hypertree width of the CRPQ; cw is a width measure for CRPQs used to specify the runtime of the algorithm proposed in [4].

The first approach, which we call here OSPG+OSYan, involves constructing an equivalent CQ Q' by materializing the results of each RPQ atom of Q, then evaluating Q' using standard approaches. The best known algorithm for materializing RPQs [5], dubbed by the authors as OSPG, runs in time $O(N + N \cdot \mathsf{OUT}_a^{1/2})$. Q' is then evaluated using an output-sensitive refinement of Yannakakis [24], called here OSYan, which is the best known output-sensitive algorithm for acyclic CQs. OSYan evaluates the CQ Q' in time $O(\mathsf{OUT}_a \cdot \mathsf{OUT}^{1-1/\text{fn-fhtw}(Q')} + \mathsf{OUT})$. Therefore, the total running time of OSPG+OSYan is $O(N + N \cdot \mathsf{OUT}_a^{1/2} + \mathsf{OUT}_a \cdot \mathsf{OUT}^{1-1/\text{fn-fhtw}(Q)} + \mathsf{OUT})$.

The second approach, which we call here PC+OSPG, is the output-sensitive algorithm for evaluating acyclic CRPQs proposed in [4]. The algorithm first transforms the original query Q into a free-connex acyclic CRPQ Q' by contracting some RPQs and "promoting" remaining bound variables into free variables. In the next step, the algorithm calibrates the RPQs of Q' with each other before invoking OSPG on each RPQ. This yields an output-sensitive algorithm for evaluating Q whose runtime depends on the number of "promoted" variables of Q. This quantity is defined in [4] as a width measure for CRPQs and is termed the "contraction width" (cw) of a CRPQ.

The following table states the evaluation times achieved by OSPG+OSYan [5, 24], PC+OSPG [4], and our algorithm for general acyclic queries.

Algorithm	Running time for acyclic CRPQs
OSPG+OSYan [5, 24]	$N + N \cdot \mathsf{OUT}_a^{1/2} + \mathsf{OUT}_a \cdot \mathsf{OUT}^{1-1/\text{fn-fhtw}(Q)} + \mathsf{OUT}$
PC+OSPG [4]	$N + N \cdot \mathsf{OUT}^{1/2} \cdot N^{\text{cw}/2} + \mathsf{OUT} \cdot N^{\text{cw}}$
Our algorithm	$N + N \cdot \mathsf{OUT}^{1-1/\max(\text{fn-fhtw}(Q),2)} + \mathsf{OUT}$

We compare these three algorithms with an example.

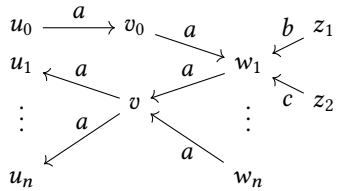


Fig. 3. The edge-labeled graph used in Example C.1

Example C.1. Consider the 3-star CRPQ $Q(X_1, X_2, X_3) = a^*aa(X_1, X) \wedge b(X_2, X) \wedge c(X_3, X)$, which asks for all triples (u, v, w) of vertices such that there is a vertex z, a path from u to z labeled by a^*aa, an edge from v to z labeled by b, and an edge from w to z labeled by c. The query is acyclic

but not free-connex. It has free-connex fractional hypertree width $w = 3$ and contraction width $\mathsf{cw} = 1$.

Consider the edge-labeled graph $G = (V_1 \cup V_2 \cup \{u_0, v_0, v, z_1, z_2\}, E_a \cup E_{bc}, \{a, b, c\})$, where $V_1 = \{u_1, \ldots, u_n\}$, $V_2 = \{w_1, \ldots, w_n\}$, $E_a = \{(w_i, a, v), (v, a, u_i) \mid i \in [n]\} \cup \{(u_0, a, v_0), (v_0, a, w_1)\}$, and $E_{bc} = \{(z_1, b, w_1), (z_2, c, w_1)\}$ for some $n \in \mathbb{N}$. The graph G is visualized in Figure 3 (left).

We observe that $|V| = 2n + 5$, $|E| = 2n + 4$, and $\mathsf{OUT}_a = \Theta(n^2)$. The output of Q consists of the single tuple (u_0, z_1, z_2), hence $\mathsf{OUT} = 1$. These quantities imply that OSPG+OSYan needs $O(n^2)$ time, PC+OSPG needs $O(n^{3/2})$ time, while our algorithm needs only $O(n)$ time to evaluate the query. □

Received December 2025; accepted February 2026