Size bounds and algorithms for conjunctive regular path queries

Tamara Cucumides
PUC Chile

Juan Reutter
PUC Chile

Domagoj Vrgoč
PUC Chile

Abstract

Conjunctive regular path queries (CRPQs) are one of the core classes of queries over graph databases. They are join intensive, inheriting their structure from the relational setting, but they also allow arbitrary length paths to connect points that are to be joined. However, despite their popularity, little is known about what are the best algorithms for processing CRPQs. We focus on worst-case optimal algorithms, which are algorithms that run in time bounded by the worst-case output size of queries, and have been recently deployed for simpler graph queries with very promising results. We show that the famous bound on the number of query results by Atserias, Grohe and Marx can be extended to CRPQs, but to obtain tight bounds one needs to work with slightly stronger cardinality profiles. We also discuss what algorithms follow from our analysis. If one pays the cost for fully materializing graph queries, then the techniques developed for conjunctive queries can be reused. If, on the other hand, one imposes constraint on the working memory of algorithms, then worst-case optimal algorithms must be adapted with care: the order of variables in which queries are processed can have striking implications on the running time of queries.

2012 ACM Subject Classification Information systems → Query languages

Keywords and phrases graph databases, regular path queries, worst-case optimal algorithms

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Graph patterns form the basis of most query languages for graph databases [1]. Consequently, there has been a lot of progress in terms of pattern query answering, either by porting and optimizing relational techniques into a graph context [16, 11, 12], or by implementing worst-case optimal algorithms over graphs, which run in time given by the AGM bound of queries [15, 10, 2], or even with a mix of both approaches [8].

However, the main focus has been so far on simple graph patterns, or *conjunctive queries* (CQs), which are matched to the queried database. But one of the key aspects that differentiate graph and relational databases is the need for answering path queries, which are usually integrated into graph patterns to form so called conjunctive regular path queries (CRPQs). CRPQs form an important use case for graph patterns [1], but so far we know little about algorithms that can compute answers of these queries.

Consider for instance the CRPQ in Figure 1. We assume in this paper the standard relational representation of graphs using one binary relation per edge label. Namely, each edge label a results in a relation R_a containing all pairs (v, v') connected by an a-labelled edge in the graph. Then Q_1 features a triple join, but one of the relations we are joining is given by expression a^+, which corresponds to the transitive closure of the relation R_a. How should one compute this query? One approach is to first *materialize* the answers of all path queries, after which we have a simple graph pattern or CQ over these materialized relations, whose answers we already know how to compute [17, 14]. In our case, this means computing the transitive closure R_a^+ of R_a, as a virtual relation, and then compute the (relational) triple join $R_a^+(x,y) \wedge R_b(y,z) \wedge R_c(z,x)$, treating now a^+ as if it was a standard relation. Is this efficient? Let us assume for simplicity that the cardinality of R_a, R_b and R_c is N. Then, the virtual relation R_a^+ may have up to N^2 tuples. If we use a worst-case algorithm for the task of computing the triple-join, we can get the answers in $O(N^2)$, which also encompass the time taken to build the virtual relation R_a^+ for dealing with a^+. As we shall see, the $O(N^2)$ bound also corresponds to the maximum number of tuples that may be in the answer of this query, so our algorithm can be dubbed *worst-case optimal*. In this case, the approach seems plausible, at least in terms of worst-case asymptotic complexity.

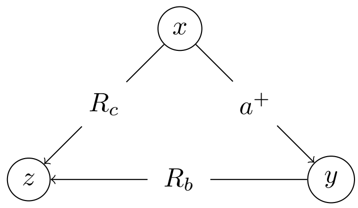

Figure 1 $Q_1(x,y,z) \leftarrow a^+(x,y) \wedge R_+(y,z) \wedge R_c(x,z)$

On the other hand, our strategy of materializing transitive closure (or more generally, any path query) can be quite costly, as R_a^+ may have up to N^2 tuples itself, which need to be stored in memory. Thus, it is natural to ask if there is any way of computing the answers for this query in an optimal way, and in such a way that we do not pay the cost of fully materializing all path queries. And perhaps more importantly, what happens with other CRPQs? Do we have a worst-case optimal algorithm for every CRPQ? Does it necessarily involve materializing all path queries beforehand?

In this paper we provide answers to these questions. We study bounds on the maximum size of the answer of a CRPQ, given certain cardinality information about the graph. We use these bounds to investigate optimal algorithms for CRPQs, either in full generality, or with additional memory constraints. Our main contributions are as follows.

1. Regarding output bounds for CRPQs, we first observe that **the bound obtained by materializing RPQs and applying the standard AGM bound on the resulting query is not tight**. For example, consider the query Q_2 in Figure 2 below:

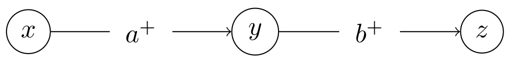

Figure 2 $Q_2(x,y,z) \leftarrow a^+(x,y) \wedge b^+(y,z)$

If $|R_a| = |R_b| = N$ then the answers to a^+ and b^+ may have up to N^2 tuples. Thus, applying the usual AGM bound over the CQ resulting from materializing both expressions into relations gives an upper bound of $O(N^4)$. This is of course not tight: since $|R_a| = |R_b| = N$, the number of possible elements in any relation is also bounded by N, so the total number of tuples in the answer is $O(N^3)$. One can show that this bound is actually tight.

2. We can obtain much more precise bounds for Q_2 if we also take into account the cardinality of the first and second components of both R_a and R_b. For example, if we assume that the cardinality of the projection of R_a and R_b over the first or second components is bounded by M, then the number of tuples in the output of Q_2 is in $O(M^3)$. And we can generalize this for every CRPQ: **We provide bounds on the number of tuples in the answer of any CRPQ, over any graph satisfying the same cardinalities of relations and each of their components**. Our upper bound is based on an extension of the linear program used to show the AGM bound. Consider for example query $Q_3(x,y,z) \leftarrow a^+(x,y) \wedge b^+(y,z) \wedge R_c(x,z)$ in Figure 3.

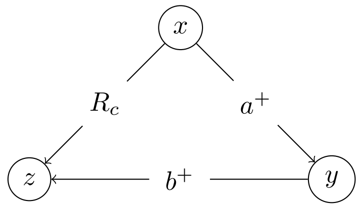

Figure 3 $Q_3(x,y,z) \leftarrow a^+(x,y) \wedge b^+(y,z) \wedge R_c(x,z)$

Let R_a^s be the projection on the first component of R_a, R_a^e the projection on the second component (and analogously for R_b). Then the answers of Q_3 over a given graph with relations R_a, R_b, R_c are bounded by 2^{ρ^*}, where ρ^* is the solution of the following program.

$$\begin{aligned}
\text{minimize} \quad & u^{R_c} \log|R_c| + u_x^{a^+} \log|R_a^s| + u_y^{a^+} \log|R_a^e| + u_y^{b^+} \log|R_b^s| + u_z^{b^+} \log|R_b^e| \\
\text{where} \quad & u^{R_c} + u_x^{a^+} \geq 1 \\
& u^{R_c} + u_z^{b^+} \geq 1 \\
& u_y^{a^+} + u_y^{b^+} \geq 1 \\
& u^{R_c}, u_x^{a^+}, u_y^{a^+}, u_y^{b^+}, u_z^{b^+} \geq 0
\end{aligned} \tag{1}$$

This is a generalization of the AGM linear program [4], in which now we can also assign weights to the starting and ending points of RPQs, which receive their own variables ($u_x^{a^+}$ and $u_y^{a^+}$ for a^+, $u_y^{b^+}$ and $u_z^{b^+}$ for b^+). Assume that the cardinality of R_a^s, R_a^e, R_b^s and R_b^e is M, and the cardinality of R_c is N, with $N \leq M^2$. Then, an optimal solution for this query is $u^{R_c} = 1$, $u_y^{a^+} = u_y^{b^+} = \frac{1}{2}$, and $u_x^{a^+} = u_z^{b^+} = 0$. Intuitively, this means assigning *full weight* to the $R_c(x,z)$ atom of the query, and evenly dividing the weights for vertex y. This makes sense, because the answers of Q are always bounded by MN: for each tuple (u,v) in R_c there are at most M nodes connected to u and v by means of the expressions a^+ and b^+.

3. Now that we know how to bound the answers of CRPQ, the next question is to look for worst-case optimal algorithms for them: an algorithm for a query Q is worst-case optimal if, on input graph G, the answers of Q over G are computed in time bounded by the maximum number of tuples in the answer of Q over any graph with the same cardinalities of all the relations as G. Unfortunately we show that, under usual complexity assumptions, **there are CRPQs for which no worst-case optimal algorithm exists.**

4. Two strategies stand off when thinking about computing the answers of CRPQs. The first we already mentioned: materialize every path query as a virtual relation, and then apply a worst case optimal algorithm such as e.g. Leapfrog Trie-join [17]. For some queries, such as the triangle query in Figure 3, this strategy appears to be as optimal as one can be, at least in terms of computation time in the worst case. However, the memory requirements are quite high, as materialized path queries can be of quadratic size in terms of the number of nodes in the graph. On the other hand, one can immediately perform Leapfrog Trie-join on the graph as if it was a relational database, and whenever one needs pairs of the form (a,x) connected by a path query r, one computes it on demand, say by doing a Breadth

First Search (BFS) over the relation. Assuming we do not cache intermediate results, this strategy has no significant memory requirements, but it may incur in chained searches on the graph, and end up being slower than materialization. At a first glance, it would appear that we have a strict time/memory tradeoff when computing this type of queries. But is this the best we can do? As it turns out, by carefully planning how RPQs are instantiated within worst case optimal algorithms, **we provide an algorithm that can compute the answers of many CRPQs under the same running time as an algorithm based on full materialization of path queries, but requiring only linear memory, in terms of the nodes of the graph**.

2 Preliminaries

Graph databases. A *graph database* is usually defined in the theoretical literature as a directed edge-labelled graph [5, 19]. More formally, if Σ is a finite alphabet of edge labels, a graph database over Σ is a pair (V, E), where V is a finite set of nodes, and $E \subseteq V \times \Sigma \times V$. An alternative way of viewing a graph database is through its relational representation. Namely, if Σ is a finite labelling alphabet, a graph database $G = (V, E)$ over Σ can be given as a relational database over the schema $\{R_a\}_{a\in\Sigma}$ of binary relations. Intuitively, $R_a(v, v')$ holds if and only if $(v, a, v') \in E$; that is, if there is an a-labelled edge between v and v'. Throughout the paper we will often switch between these two representations. For a binary relation R_a, with $a \in \Sigma$, we denote with R^s_a the projection of R_a onto its first attribute. Similarly we define R^e_a as the projection of of R_a onto the second attribute.

Queries over graph databases. Path queries are usually given as regular expressions, under the name of Regular Path Queries, or RPQs. An RPQ r selects, in a graph G, all pairs (u, v) of nodes that are connected via edge labels forming a word in the language of r. We denote this set of pairs as $[\![r]\!]_G$, see Table 1 for the definition. We assume RPQs are given both by regular expressions or automata, and freely switch between these representations.

Conjunctive regular path queries (CRPQs) [1, 5], are simply conjunctions of path queries. In order to exploit what is known about size bounds for relational CQs, we separate the expressions in our CRPQ into two sets: (i) the expressions consisting of a single letter (which are thus equivalent to an ordinary CQ); and (ii) regular expressions whose languages contain more than a single letter. Therefore, we define a conjunctive regular path query over a graph database to be given by an expression

$$Q(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i) \tag{2}$$

where $a_i \in \Sigma$, r_i is a regular expression whose language is not equal to a single one letter word over Σ, and $\overline{x} = \{x_1, \ldots, x_n\} \subseteq \{y_1, z_1, \ldots, y_k, z_k\}$ is a set of output variables. A CRPQ without such regular expressions is simply a *conjuntive query* (CQ). Further, a CRPQ is *full* if every variable y_i, z_i is also mentioned in $\bar{x}$, and it is ε-free is none of the expressions r_i admit ε in their language. The expression to the right of the arrow is the *body* of query Q.

The semantics of a CRPQ Q, over a graph G is given via homomorphisms [1]. Namely, a mapping $\mu : \{x_1, \ldots, x_n\} \to V$ is an output of Q over G when μ can be extended to the variables of Q in such a way that for each $i \in \{1, \ldots \ell\}$ $R_{a_i}(\mu(y_i), \mu(z_i))$ holds, and for each $i \in \{\ell + 1, \ldots k\}$, $(\mu(y_i), \mu(z_i)) \in [\![r_i]\!]_G$. We denote the set of all outputs with $\text{Eval}(Q, G)$. A CRPQ Q is compatible with a graph G if the graph features all relations mentioned in Q.

Cardinality Profiles. For a given graph G, we use r^s to denote the number of elements in G that can participate as starting elements for a path labelled by r in G: it corresponds

$$\begin{array}{rclrcl}
[\![\varepsilon]\!]_G & = & \{(u,u) \mid u \text{ is a node id in } G\} & [\![a]\!]_G & = & \{(u,v) \mid (u,a,v) \in G\} \\
[\![r_1 \cdot r_2]\!]_G & = & [\![r_1]\!]_G \circ [\![r_2]\!]_G & [\![r_1 + r_2]\!]_G & = & [\![r_1]\!]_G \cup [\![r_2]\!]_G \\
[\![r^+]\!]_G & = & [\![r]\!]_G \cup [\![r \circ r]\!]_G \cup [\![r \circ r \circ r]\!]_G \cup \cdots & [\![r^*]\!]_G & = & [\![\varepsilon]\!]_G \cup [\![r^+]\!]_G
\end{array}$$

Table 1 Semantics of RPQs, for $a \in \Sigma$, and r, r_1 and r_2 arbitrary RPQs. The symbol $\circ$ denotes the composition of binary relations.

to the union of each R^s of each relation R that labels a transition out of the initial state of the automaton for r. Likewise, r^e is the union of each R^e of each relation that labels a transition into a final state of the automaton for r.

In order to reason about bounds on graph databases, we always assume access to some basic statistics about the size of relations in the graph. Formally, the *cardinality profile* of a graph G over Σ with respect to a query Q includes the following cardinalities:

- $|V|$ the total number of nodes;
- For each atom $R_a(y,z)$ in Q, the number $|R_a|$ of a-labelled edges;
- For each atom $r(y,z)$ in Q, with r a regular expression, the number r^s of starting elements and r^e of final elements participating in r.

To avoid extra notation, we also assume that graphs G have access to every unary relation of the form r^s or r^e. Notice one can always add these unary relations in linear time.

The AGM bound. Atserias, Grohe and Marx [4] link the size bound of a relational join query to the optimal solution to a given linear program. In graph terms, let $Q(x_1,\ldots,x_n) \leftarrow \bigwedge R_{a_i}(y_i,z_i)$ be a full conjunctive query without self-joins, i.e, in which each a_i is different, and let G be a graph database where the size of each R_{a_i} is N_i. Atserias et. al. [4] show that an optimal bound is achieved by considering the following linear program:

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{i=1}^{n} u^{R_{a_i}} \; logN_i & \\
\text{where} & \displaystyle\sum_{\text{i : } x \text{ appears in atom } R_{a_i}} u^{R_{a_i}} \geq 1 & \text{for each variable } x \text{ in } Q \\
& u^{R_{a_i}} \geq 0 & \text{for } i = 1,\ldots,m
\end{array} \tag{3}$$

Let us denote by $\rho^*(Q,D)$ the optimal value of $\sum_{i=1}^{n} u^{R_{a_i}} \; \log N_i$. The AGM bound [4] can then be stated as follows.

▶ **Theorem 1** (AGM bound). *Let Q be a full CQ without self joins, D a database instance and $\rho^*(Q,D)$ the optimal solution of the associated linear program (3). Then,*

$$|Eval(Q,D)| \leq 2^{\rho^*(Q,D)}.$$

Furthermore, there are arbitrary large instances D for which we have $|Eval(Q,D)| = 2^{\rho^*(Q,D)}$.

We remark that all the results in this paper refer to *data complexity*, and thus the size of CRPQs is treated as a constant thorough our analysis.

3 Size bounds for CRPQs

Path queries provide an interesting challenge when studying size bounds. Every path query is a relation in itself, but in the worst case, a query like $a^+(x,y)$ may end up connecting all elements in R_a^s with all elements in R_a^e, thus invoking a quadratic jump in terms of the size of the potential vertices matching to x and to y. For this reason, tight output bounds must take into account the number nodes that can participate as the starting point and the ending point of the expressions mentioned in the queries. We show how to construct a modified linear program, extending that of [4], that we use to provide our size bounds.

3.1 Motivation: underlying flat CQs

To see the intuition for our linear program, let us come back to query $Q_3(x, y, z)$ in Figure 3, and consider a graph G. In order to bound the size of $\text{Eval}(Q_3, G)$, we reason in terms of the size of $[\![a^+]\!]_G$. In the worst possible case, we have that $[\![a^+]\!]_G = R_a^s \times R_a^e$, that is, any node from R_a^e can be reached from any node from R_a^s. It is then easy to see that the answers in the evaluation $\text{Eval}(Q_3, G)$ will always be contained in what we call the *flat* CQ

$$\mathit{flat}(Q_3)(x, y, z) \;\leftarrow\; R_a^s(x) \wedge R_a^e(y) \wedge R_b^s(y) \wedge R_b^e(z) \wedge R_c(x, z),$$

in which every path query is replaced by the cross product of two unary relations, the possible starting nodes and the possible ending nodes. In fact, assuming each of R_a^s, R_a^e, R_b^s and R_b^e are unary relations in G, we have that $|\text{Eval}(Q_3, G)| \leq |\text{Eval}(\mathit{flat}(Q_3), G)|$, and this hold for any graph G compatible with Q. Now $\mathit{flat}(Q_3)$ is a full CQ without self-joins, and we know how to bound its output [4], which immediately results in an upper bound for Q_3.

Interestingly, the focus on flat conjunctive queries has another intuitive reading. Coming back to query Q_2 from Figure 2, its flat version is simply a cross product of unary relations

$$Q_2(x, y, z) \;\leftarrow\; R_a^s(x) \wedge R_a^e(y) \wedge R_b^s(y) \wedge R_b^e(z).$$

For a graph G in which all of R_a^s, R_a^e, R_b^s and R_b^e have N nodes, we verify that $|\text{Eval}(Q, G)| \leq N^3$. This cubic bound is, in a sense, the most crude upper bound one could get for a conjunctive query: it is simply the cross product of every vertex matching for x, y and z. It just happens that when the labels joining x and y, and y and z are path queries, this crude bound ends up being realistic.

But is it tight? We can show it is, and our size bounds end up enjoying several good properties proved before for full relational join queries [4] or conjunctive queries [9]. Moving from this simple example to arbitrary CRPQs, however, is not that easy, and we proceed in several steps. In section 3.2 we start with a fragment of CRPQs for which the proof is simpler, and the bounds much more elegant. This fragment corresponds to full CRPQs, without self-joins or any repetition of labels between atoms, and whose RPQs are defined by ε-free expressions that admit at least one word of length 2. We call this fragment *Simple* CRPQs, and the reason for starting with this fragment is that we can define the general upper and lower bounds exactly in the same way they were defined for simple CQs by Atserias et al. in their seminal paper[4]. We then extend our results to arbitrary CRPQs defined by ε-free expressions, with the only caveat that our lower bound is now up to a constant that depends on the query. We finish with CRPQs that may use expressions including ε, such as a^*, which is one of the most common path query occurring in practice [6]. We deal with them by separating into ε and ε-free parts, which we can then treat independently.

3.2 Simple CRPQs

To state our first result, we provide a formal definition of the aforementioned simple fragment. A *simple* CRPQ is a *full* CRPQ of the form $Q(\overline{x}) \;\leftarrow\; \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ with the following properties:

- Each relation R_{a_i} appears only once in Q (no self joins);
- All regular expressions r_i are ε-free and they contain a word of length at least 2;
- If r and r' are two different regular expressions in Q, then no endpoint label of r (i.a. a label of a transition going out of the initial state of the automaton for r, or going into the final state of the automaton for r) is an endpoint label for r'.

As we hinted in the introduction, the idea is to extend the linear program of AGM with one *vertex* variable for each endpoint of every atom $r(x, y)$ in the query, which are then constrained in the same fashion as edge variables. Alternatively, one can directly construct the program for the corresponding flat query: it happens to be exactly the same program.

▶ **Theorem 2** (Bound for simple CRPQs). *Assume that the query* $Q(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ *is a simple CRPQ. Then for any graph G we have that*

$$|Eval(Q, G))| \leq 2^{\rho^*(Q,G)}$$

where $\rho^*(Q, G)$ *is the optimal solution of the following linear program:*

$$\begin{aligned}
&\textit{minimize} && \sum_{i=1}^{\ell} u^{R_{a_i}} \log |R_{a_i}| + \sum_{i=\ell+1}^{k} (u^{r_i}_{y_i} \log |r^s_i| + u^{r_i}_{z_i} \log |r^e_i|) \\
&\textit{where} && \sum_{i:x=y_i \vee i:x=z_i} u^{R_{a_i}} + \sum_{i:x=y_i} u^{r_i}_{y_i} + \sum_{i:x=z_i} u^{r_i}_{z_i} \geq 1 \quad \textit{for } x \in \overline{x} \\
& && u^{R_{a_i}} \geq 0 \quad \textit{for } i \in [1, \ell] \\
& && u^{r_i}_{y_i}, u^{r_i}_{z_i} \geq 0 \quad \textit{for } i \in [\ell+1, k]
\end{aligned} \tag{4}$$

Furthermore there are arbitrarily large instances for which

$$|Eval(Q, G))| \geq 2^{\rho^*(Q,G)}.$$

The upper bound. For completeness, in the Appendix we give a direct proof of this Theorem, but we can obtain a simple upper bound by using flat CQs. Let $Q(\overline{x})$ be a simple CRPQ. Its underlying *flat* query *flat*(Q) is the conjunctive query defined as:

$$\textit{flat}(Q)(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r^s_i(y_i) \wedge r^e_i(z_i)$$

Recall we assume for simplicity that each r^s and r^e is an unary predicate already present in G. The following is now easy to check:

▶ **Lemma 3.** *Eval*$(Q, G) \subseteq$ *Eval*$($*flat*$(Q), G)$*, with Q a simple CRPQ.*

Since the linear programs of both *flat*(Q) (as in [4]) and Q (as in the statement of Theorem 2) coincide, and $2^{\rho^*(\textit{flat}(Q),G)}$ is an upper bound for Eval$($*flat*$(Q), G)$, this immediately proves the upper bound of Theorem 2.

The lower bound. We will prove the lower bound by constructing an instance out of the dual program for Q. Let us first illustrate the tightness of the bound via the means of an example. Consider again query $Q_3(x, y, z) \leftarrow a^+(x, y), b^+(y, z), R_c(x, z)$.

The linear program for this query is as seen in (1) and the corresponding dual is:

maximize: $v_x + v_y + v_z$

subject to: $v_x + v_z \leq \log |R_c|$

$$v_x \leq \log |(a^+)^s| \qquad v_y \leq \log |(a^+)^e|$$
$$v_y \leq \log |(b^+)^s| \qquad v_z \leq \log |(b^+)^e|$$
$$v_x, v_y, v_z \geq 0$$

Consider an optimal solution for the primal $\overline{x}$ and (for duality) a solution to the dual (v_x, v_y, v_z) such that $\rho^*(Q, D) = v_x + v_y + v_z$. Now we want to build an instance G such that $\text{Eval}(Q, G) = 2^{\rho^*(Q,G)}$ with cardinalities as above. The instance is defined as follows,

- We have a special vertex $\star$ and 3 sets of vertices: $|V_x| = 2^{v_x}$, $|V_y| = 2^{v_y}$, $|V_z| = 2^{v_z}$ such that $V_x \cap V_y \cap V_z = \{\star\}$
- Add edges (x, c, z) for every pair of nodes $(x, z) \in V_x \times V_z$
- Add edges $(x, a, \star)$ for every $x \in V_x$ and edges $(\star, a, y)$ for $y \in V_y$
- Finally, add edges $(y, b, \star)$ for $y \in V_y$ and $(\star, b, z)$ for $z \in V_z$.

By the dual restrictions, we can check that the cardinalities are equal or smaller than we wanted (if they're smaller we can add random edges as this can only increase the number of tuples of $\text{Eval}(Q, G)$)). Also we can check that $|\text{Eval}(Q, G))| = 2^{v_x+v_y+v_z}$ since we have all tuples (x, y, z) with $x \in V_x$, $y \in V_y$ and $z \in V_z$. We conclude that $|\text{Eval}(Q, G))| = 2^{\rho^*(Q,G)}$.

Now we formalize this construction for any simple CRPQ:

Proof of Theorem 2, lower bound. As before, we use the dual program of equation (4)

$$
\begin{aligned}
\text{maximize: } & \sum_{x \in \overline{x}} v_x \\
\text{subject to: } & v_{y_i} + v_{z_i} \leq \log |R_{a_j}|, && i = 1, \ldots, \ell \\
& v_{y_i} \leq \log |r_j^s|, && i = \ell+1, \ldots, k \\
& v_{z_i} \leq \log |r_j^e|, && i = \ell+1, \ldots, k \\
& v_x \geq 0, && x \in \overline{x}
\end{aligned}
$$

Consider an instance with cardinalities $|R_{a_i}| = N_i$ for $i \in [1, \ell]$, $|r_j^s| = N_j^s$ and $|r_j^e| = N_j^e$ for $j \in [\ell+1, k]$. By duality, for any solution $\overline{u}$ to the primal and $\overline{v}$ for the dual, we have that

$$\sum_{i=1}^{\ell} u^{R_{a_i}} \log |R_{a_i}| + \sum_{i=\ell+1}^{k} (u_{y_i}^{r_i} \log |r_i^s| + u_{z_i}^{r_i} \log |r_i^e|) \geq \sum_{x \in \overline{x}} v_x,$$

with equality when the solutions are optimal. Let us assume that all N_i, N_i^s and N_i^e are of the form 2^{L_i} for some $L_i \in \mathbb{N}$ so the optimal solution of both the primal and dual are rational. Let $\overline{v}$ be the dual solution and write each v_x as p_x/q. Then $\overline{p}$ is an optimal solution to the linear program with cardinalities N_i^q. Now we present a graph database G with $|R_i| = N_i^q$, $|r_i^s| = (N_i^s)^q$ and $|r_i^e| = (N_i^e)^q$ such that $|\text{Eval}(Q, G)| \geq 2^{\rho^*(Q,G)}$.

- The vertices of G is the union of sets $V_x = \{1, \ldots, 2^{v_x}\}$ for each $x \in \overline{x}$. Also consider a vertex $\star$ that is part of every V_x.
- For every atom $R_{a_i}(y_i, z_i)$ in Q, add to G one edge (u, a_i, v) for every pair (u, v) in $V_{y_i} \times V_{z_i}$.
- For every atom $r_i(y_i, z_i)$ in Q, choose an arbitrary word $\pi_i = a_{i_1} \ldots a_{i_N}$ of length at least 2 in the language of r_i and
  - Add to G the edges $(u, a_{i_1}, \star)$ for each $u \in V_{y_i}$.
  - Add to G edges $(\star, a_{i_j} \star)$ for every $j \in [2, N-1]$.
  - Add to G the edges $(\star, a_{i_N}, v)$ for each $v \in V_{z_i}$.

From the construction we verify that:

$$
\begin{aligned}
|R_{a_i}| = 2^{v_{y_i}+v_{z_i}} && \leq 2^{q \log N_i} = N_i^q && \forall i \in [1, \ell] \\
|r_i^s| = 2^{v_{y_i}} && \leq 2^{q \log N_i^s} = (N_i^s)^q && \forall i \in [\ell+1, k] \\
|r_i^e| = 2^{v_{z_i}} && \leq 2^{q \log N_i^e} = (N_i^e)^q && \forall i \in [\ell+1, k]
\end{aligned}
$$

Further, we also verify that $\text{Eval}(Q,G)$ contains all tuples $t \in V_{x_1} \times \cdots \times V_{x_n}$. Now we add random edges and vertices such that $|R_i| = N_i^q$, $|r_i^s| = (N_i^s)^q$ and $|r_i^e| = (N_i^e)^q$. We now have a graph G with the desired cardinality profile for which:

$$|\text{Eval}(Q,G)| \geq \prod_{i=1}^{l} |R_{a_i}|^{u^{R_{a_i}}} \prod_{i=l+1}^{m} |r_i^s|^{u_{y_i}^{r_i}} \cdot |r_i^e|^{u_{z_i}^{r_i}} = 2^{\sum_{x\in\overline{x}} v_x}$$

◀

As in Atserias et al., we also show that the instances satisfying the lower bound can be constructed with a certain degree of regularity, in which all cardinalities are equal.

▶ **Corollary 4.** *Given a simple CRPQ Q, we can build an arbitrarily large instance G such that $|Eval(Q,G))| \geq 2^{\rho^*(Q,G)}$ with $|R_{a_i}| = |r_j^s| = |r_j^e|$ for every relation i and j such that $u^{R_{a_i}} > 0$, $u_{y_j}^{r_j} > 0$ and $u_{z_j}^{r_j} > 0$.*

Unfortunately, not every combination of cardinalities of relations and vertices can be shown to produce tight bounds. However, as in [4], we can show the following: Let Q be a simple CRPQ and G a graph. Then there exists a graph G' with the same cardinalities as G in all vertices and relations mentioned in Q, such that $\text{Eval}(Q,G')| \geq 2^{\rho^*(Q,G)-n}$, where n is the number of attributes of Q. As for CQs, this is essentially the best we can get.

3.3 Bound for arbitrary ε-free CRPQs

Gottlob et al. study how to go from relational join queries to CQs [9], and the same techniques can be used for obtaining size bounds for ε-free CRPQs, even if they feature projections, repetition of variables, or expressions allowing only words of size 1. Bounds remain tight, except this time they are tight up to a factor that does depend on the query (but not the data) in a polynomial way. We first show how to handle arbitrary full CRPQs that are ε-free (and not just the simple ones), and then move to CRPQs that project out some variables.

From full to simple CRPQs. We first show that for a full CRPQ Q that is also ε-free, and a graph database G compatible with Q, we can construct a simple CRPQ Q', and an instance G' compatible with Q' such that $\text{Eval}(Q,G) = \text{Eval}(Q',G')$. The translation from Q to Q' has to deal with repeated labels/relations, and also with expressions that accept only words of length 1. For this, we first, replace every appearance of a relation R_a or label a in any atom of Q with a fresh relation or label not used elsewhere in the query. Next, replace any atom of the form $r_i(y_i,z_i)$ where $r_i = (a_1|a_2|\dots|a_k)$ (i.e. an expression accepting only words of size 1), with an atom $R_{r_i}(y_i,z_i)$, where R_{r_i} is a fresh relation. Translation from a graph G compatible with Q, to a graph G' compatible with Q' is constructed by assigning every copy of R_a (introduced in the construction of Q') the same tuples as R_a, and by assigning to R_r, for an expression $r = (a_1|a_2|\dots|a_k)$, the tuples in the union of all $R_{a_1},\dots,R_{a_k}$. Other relations are the same as in G.

▶ **Proposition 5** (full CRPQs). *Consider a full CRPQ of form (2) in which every r_i is ε-free. For this query we have that $|Eval(Q,G)| = |Eval(Q',G')| \leq 2^{\rho^*(Q',G')}$, where Q' and G' are constructed as above. Furthermore, one can construct arbitrarily large instances G such that $|Eval(Q,G)|2^{p(|Q|)} \geq 2^{\rho^*(Q',G')}$ where $p(|Q|)$ is a polynomial that depends exclusively on Q.*

Bounds for projections of full, ε-free CRPQs. Consider a (non-full) ε-free CRPQ of the form

$$P(\overline{x}_0) \ \leftarrow \ Q(\overline{x}), \qquad (5)$$

with $\overline{x}_0 \subsetneq \overline{x}$, and where Q is full and ε-free. From our previous result, we know that $\mathrm{Eval}(Q,G)$ is always bounded by $2^{\rho^*(Q',G')}$, where Q' and G' constructed as above. As in [9], we consider a relaxation of the linear program for Q', in which we only keep those restrictions that refer to variables of Q (and Q') that are in $\overline{x_0}$. Formally, we denote by $2^{\rho^*_{\overline{x}_0}(Q',G')}$ the optimal solution of a modified linear program for Q' and G', where in the restrictions of (4) we only consider those referring to $\overline{x_0}$. We then have:

▶ **Proposition 6** (Queries with projections [9]). *Given an CRPQ P of the form (5) then for every graph database instance G we have that $|Eval(P,G)| \leq 2^{\rho^*_{\overline{x}_0}(Q',G')}$. Moreover, there are arbitrarily large instances G such that $|Eval(P,G)|2^{p(|Q|)} \geq 2^{\rho^*_{\overline{x}_0}(Q',G')}$, where $p(|P|)$ is a polynomial that depends exclusively on P.*

3.4 Dealing with ε

As we have mentioned, the evaluation of the expression ε over a graph $G = (V,E)$ contains the *diagonal* $D = \{(v,v) \mid v \in V\}$. Thus, the evaluation of expressions containing ε, such as a^*, are somehow the union of two different sets of results. One one hand there is the ε-free part, that we know how to deal with, and on the other there is ε, which behaves more like a relation, albeit drawing pairs only from the diagonal D.

The expression ε. Consider the triangle query $Q_4(x,y,z) \leftarrow R_a(x,y) \wedge R_b(y,z) \wedge \varepsilon(x,z)$, featuring two edge labels and the regular expression ε. One can check that Q_4 is equivalent to $R_a(x,y) \wedge R_b(y,z) \wedge \varepsilon^s(x) \wedge \varepsilon^e(z) \wedge x = z$. What we have done is to produce an analogue of the *flat* version of CRPQs, and we use the equalities to force the flat part to map only to the diagonal. We further transform this query by noting that $\varepsilon^s = \varepsilon^e = V$, and *chasing* away the equality, obtain the query $R_a(x,y) \wedge R_b(y,x) \wedge V(x)$, which always produces the same number of tuples as Q_4. Hence, dealing with epsilon involves (1) transforming every atom $\varepsilon(x,y)$ into two unary atoms $V(x), V(y)$ (to be interpreted as V), plus the corresponding equality $x = y$, and (2) chasing away such equalities. It is not difficult to see that both of these operations do not alter the size of the outputs of queries; the transformation always yields an equivalent query, save for the case when the arity of the query is reduced when chasing the equalities.

Formally, assume that Q is a CRPQ, and let $Q^{\setminus\varepsilon}$ be the query in which each atom $\varepsilon(x,y)$ is replaced for the construct $V(x) \wedge V(y) \wedge x = y$. Assuming V is interpreted as the set of vertices in every graph $G = (V,E)$, we have:

▶ **Lemma 7.** *For every CRPQ Q and graph G, $Eval(Q,G) = Eval(Q^{\setminus\varepsilon},G)$*

Further, let Q be a CRPQ with equalities, i.e, additional atoms of the form $x = y$, where both x and y appear in a non-equality atom in Q. Let $\mathrm{chase}(Q)$ be CRPQ resulting by repeatedly replacing variable y for variable x for each atom $x = y$ in the query. We have:

▶ **Lemma 8.** *For every CRPQ Q and graph G, $|Eval(Q,G)| = |Eval(chase(Q),G)|$*

In order to formally state the bound for queries with ε, we use again query Q' and graph G' constructed in the previous subsection, as well as the solution $2^{\rho^*_{\overline{x}_0}(Q',G')}$ for the modified linear program for Q' and G'.

▶ **Proposition 9.** *Let $P(\overline{x_0})$ be a CRPQ in which every regular expression is either ε, or is ε-free, and G a graph, and assume that the body of $chase(P^{\setminus\varepsilon})$ is of the form $Q(\overline{x})$, with $\overline{x_0} \subseteq \overline{x}$. Then for every graph database instance G we have that $|Eval(P,G)| \leq 2^{\rho^*_{\overline{x}_0}(Q',G')}$. Moreover, there are arbitrarily large instances G such that $|Eval(P,G)|2^{p(|P|)} \geq 2^{\rho^*_{\overline{x}_0}(Q',G')}$, where $p(|P|)$ is a polynomial that depends exclusively on P.*

Arbitrary RPQs. Arbitrary RPQs such as a^* are not so easy to deal with, as they represent, somehow, the union of the diagonal database and an ε-free CRPQ. Consequently, we will look into *splitting* CRPQs into parts with ε and parts without it. For a given CRPQ Q, let $r_{\ell_1}, \ldots, r_{\ell_p}$ be the RPQs in Q that accept ε. We define the family of queries $Q[S]$, for $S \subseteq \{\ell_1, \ldots, \ell_p\}$, as follows. For each $r_\ell, \ell \in \{\ell_1, \ldots, \ell_p\}$, find a decomposition $r_\ell = \varepsilon + \hat{r}_\ell$, where $\hat{r}_\ell$ is ε-free. Then atom $r_\ell(y_\ell, z_\ell)$ is replaced by $\hat{r}_\ell(y_\ell, z_\ell)$, if $\ell \in S$, or by $K_\ell(y_\ell) \wedge K_\ell(z_\ell) \wedge y_\ell = z_\ell$, where K_ℓ is a fresh relation symbol, if $\ell \notin S$.

Now augment any graph G to make it compatible with any $Q[S]$ by adding relation $K_\ell = \{a \mid a \notin \hat{r}^s_\ell \cap \hat{r}^e_\ell\}$ for each $\ell \in \{\ell_1, \ldots, \ell_p\}$. It is not too difficult to prove that $|\mathrm{Eval}(Q, G)| \leq \sum_{S \subseteq \{\ell_1, \ldots, \ell_p\}} |\mathrm{Eval}(\mathit{flat}(Q[S]), G)|$, and we can further turn this property into an output bound for queries[1].

▶ **Proposition 10.** *Let Q be a CRPQ. For any graph G we have that $|Eval(Q, G)| \leq \sum_{S \subseteq \{\ell_1, \ldots, \ell_p\}} 2^{\hat{\rho}^*(Q[S], G)}$., where queries $Q[S]$ are defined as above, and $2^{\hat{\rho}^*(Q[S], G)}$ is the size output bound shown for $Q[S]$, in Proposition 9. Moreover, there are arbitrarily large graphs for which this bound is tight.*

One important caveat of this results, is that the instances showing that the bound is tight work by constructing graphs G in which, for every expression $r_\ell = \varepsilon + \hat{r}_\ell$, we verify that $[\![\varepsilon]\!]_G \subseteq [\![\hat{r}_\ell]\!]_G$.

4 WCO algorithms for CRPQs

In this section we deal with algorithms for computing CRPQs. Ideally, one would expect an algorithm that runs in the worst-case optimal bound from Theorem 2 (and subsequent generalizations). We call such an algorithm worst-case optimal, or wco algorithm for short. Unfortunately, bounds from Casel and Schmid [7] directly imply that such algorithms do not exist under usual complexity assumptions. In the light of this, we establish a baseline which amounts to first computing all the answers to the regular expressions mentioned in our query, materializing them, and running a classical wco algorithm (e.g. GenericJoin [14]) on these materialized relations. We show that a modification of the GenericJoin algorithm of [14] can approach the optimal performance of our baseline for many CRPQs. As is usual in algorithms for relational/graph queries, we will assume all our queries to be full.

4.1 WCO algorithms for CRPQs may not exist

Casel and Schmid show lower bounds for the problem of evaluating a single RPQ [7]. Specifically, for a graph $G = (V, E)$, and a (regular path) query $Q(x, y) \leftarrow r(x, y)$, they prove that any algorithm capable of evaluating Q over G in time $O(|V|^\omega f(|Q|))$ can also be used to solve the *Boolean Matrix Multiplication* (BMM) problem: given two square matrices A and B of size n, compute the product matrix $A \times B$, in time $O(n^\omega)$. In particular, this means that a quadratic algorithm for computing path queries does not exist unless the BMM hypothesis is false, and if we accept the weaker *combinatorial* BMM hypothesis [18], then no subcubic algorithm exists for computing Q. Since the answers to Q are clearly bounded by $|V|^2$, then we cannot hope for a worst-case optimal algorithm in this case.

A natural question is what happens with CRPQs that mix both path queries and relations in their edges. Perhaps the relations help soften the underlying complexity of the

[1] For CRPQs with equalities, *flat*(Q) is defined just as before, all equalities are maintained.

problem? Unfortunately, this is not the case. To see this, consider query $Q(x, y, z) \leftarrow R_a(x, y) \wedge S_b(y, z) \wedge r(x, z)$, where r is any regular expression. Given a graph G in which $|R_a| = |S_b| = n$, our results tell us that the answer of Q over G contains at most $O(n^2)$ tuples, and thus a worst-case algorithm must evaluate Q in time $O(n^2)$. But this algorithm can then be used to compute the answers for r over a graph $G = (V_G, E_G)$, where V_G contains at least n nodes $v_1, \ldots, v_n$. For this, we construct a graph database $G' = (V_G \cup \{1\}, E_{G'})$, where $R_a = \{(v_i, 1) \mid 1 \leq i \leq n\}$, $S_b = \{(1, v_i) \mid 1 \leq i \leq n\}$ and where the rest of the relations are as in G. Then a tuple $(v_i, 1, v_j)$ is in $\mathrm{Eval}(Q, G')$ if and only if (v_i, v_j) is an answer to r on G.

▶ **Proposition 11.** *An algorithm capable of computing the answers of every simple CRPQ Q over a graph G in time $O(2^{\rho^*(Q,G)})$ refutes the BMM hypothesis.*

Having ruled out the possibility of worst-case optimal algorithms, let us review what can we do with existing techniques.

As our baseline, we establish a rather naive algorithm, called FullMaterialization, which evaluates a CRPQ Q over a graph database G as follows:

1. Compute the answer of each RPQ r appearing in Q over G.
2. Materialize all of these binary relations and add them to G.
3. Use a (relational) wco algorithm (e.g. GenericJoin [14]) to compute the query answer.

In the final step, each RPQ is now simply treated as a relation that we have previously computed. This algorithm runs in time bounded by the time to compute the RPQs from Q, and the AGM bound of the query. However, the algorithm may require memory that is quadratic in terms of the nodes in the graph, to be able to store the results of RPQs.

While reasonable, this algorithm has practical issues: the quadratic memory footprint may be too big to store in memory, and we may be performing useless computations because most pairs in the answers of RPQs may not even match to the remainder of patterns. Memory usage may be alleviated by clever usage of compact data structures, as in e.g. [3], but we take a different approach.

In what follows, we impose that algorithms may only use $O(|V|)$ memory, for $G = (V, E)$. Since Proposition 11 rules out strict wco algorithms, our goal is to devise algorithms that are capable of achieving the running time of FullMaterialization, but using just linear memory (in data complexity). To analyse the running time of the algorithm, we first introduce some notation. For a CRPQ Q and a graph database G, with $\mathrm{AGM}(Q, G)$ we denote the bound for maximal size of $\mathrm{Eval}(Q, G')$, over all graphs G' that have the same cardinality profiles as G (this includes both the cardinalities of all the relations, as well as the projections on starting and ending points of these). The time complexity of FullMaterialization for a query Q, over a graph $G = (V, E)$, is bounded by $O(|V|^3 + \mathrm{AGM}(Q, G))$, where the cubic factor accounts to materializing all the RPQs in Q.

4.2 GenericJoin for CRPQs

In order to avoid materializing relations which are potentially quadratic in the size of the graph, we can utilize a simple idea: compute RPQs on-demand, the first time such an answer is needed. For this, we will adapt the (relational) wco algorithm GenericJoin of [14], so that it processes regular relations as needed. As we will see, this approach gives us good running time even when the memory is constrained, and can actually run under the FullMaterialization time bounds for a broad class of queries. For CRPQs, however, the order of variables we work with has striking implications on the efficiency of the algorithm.

If $Q(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ is a full CRPQ, and G a graph database, then Algorithm 1 defines GenericJoinCRPQ(Q, G), a generalization of the GenericJoin

Algorithm 1 GenericJoinCRPQ(Q, G)

1: ▷ Q May have unary relations of the form $r[v]$, from previous recursive iterations.
2: $A \leftarrow \emptyset$
3: **if** $|\overline{x}| = 1$ **then**
4:     **return** Eval(Q, G)
5: Pick a variable $x \in \overline{x}$ ▷ We compute into L nodes that can potentially map to x
6:
$$L \leftarrow \bigcap_{R(x,z)\in Q} R^s \bigcap_{R(y,x)\in Q} R^e \bigcap_{r(x,z)\in Q} r^s \bigcap_{r(y,x)\in Q} r^e \bigcap_{r[v](x)\in Q} r[v]$$
7: **for** $v \in L$ **do**
8:     $\hat{Q} \leftarrow Q[x/v]$, $\hat{G} \leftarrow G$ ▷ We instantiate x to v in $\hat{Q}$
9:     **for** each atom $r(v, z) \in \hat{Q}$ **do** ▷ Compute answers to $r(v, z)$, store them in $r[v](z)$
10:         $\hat{G} \leftarrow G \cup r[v]$, with $r[v] = \{v' \mid (v, v') \in [\![r]\!]_G\}$
11:         replace $r(v, z)$ for $r[v](z)$ in $\hat{Q}$
12:     **for** each atom $r(y, v) \in \hat{Q}$ **do** ▷ Compute answers to $r(y, v)$, store them in $r[v](y)$
13:         $\hat{G} \leftarrow G \cup r[v]$, with $r[v] = \{v' \mid (v', v) \in [\![r]\!]_G\}$
14:         replace $r(y, v)$ for $r[v](y)$ in $\hat{Q}$
15:     $A[v] \leftarrow$ GenericJoinCRPQ$(\hat{Q}, \hat{G})$
16:     $A \leftarrow A \cup \{v\} \times A[v]$
17: **return** A

wco algorithm from the relational setting to graphs and (full) CRPQs. Similarly as in [14], we assume an order on the variables of Q, and start to recursively strip one variable at a time. For a selected variable, we compute all the nodes that can be bound to this variable (line 5). Then we iterate over these nodes one by one, compute RPQs as needed, adding them to the database (lines 9–11 and 12–14), and proceed recursively (line 15). For the base case when we have only one variable, we simply complete the missing values (line 4).

Analysis. So how does this algorithm compare to FullMaterialization? Well, this is heavily dependent on the CRPQ we are processing. As an example, consider again the triangle query with two RPQs, $Q_3(x, y, z) \leftarrow a^+(x, y) \wedge b^+(y, z) \wedge R_c(x, z)$ as in Figure 3, and consider a graph G in which $|R_c| = N$ and all starting and ending points of RPQs a^+ and b^+ have cardinality M. Here FullMaterialization runs in time $O(M^3 + MN)$, but with quadratic memory (the first part of the sum is for computing answers of RPQs, the second part is the max number of outputs of the query). On the other hand, GenericJoinCRPQ achieves the same bound, but using only linear memory. To see this, let us assume the first chosen variable is y. As per line 5, we first iterate over all possible vertices v in $L = {b^+}s \cap {a^+}^e$. For each such value, we compute sets $a^+[v] = \{v' \mid (v', v) \in [\![a^+]\!]_G\}$ and $b^+[v] = \{v' \mid (v, v') \in [\![b^+]\!]_G\}$, storing these in memory and adding them to G (here $\hat{G}$ is the augmented graph storing these relations). We then process the query $\hat{Q}(x, v, z) \leftarrow a^+[v](x) \wedge b^+[v](z) \wedge R_c(x, z)$ over the augmented graph $\hat{G}$. This query does not feature regular expressions, so we can compute its answers using GenericJoin$(\hat{Q}, \hat{G})$ from [14]. Further, the AGM bound for $\hat{Q}(x, v, z)$ is N (the query is *acyclic*), so the algorithm computes the answers in $O(N)$. Thus, the total running time is in $O(|L| \cdot (M^2 + N)) = O(M \cdot (M^2 + N))$. Again, the first part of the sum is for computing the answers of the path queries, the second part for evaluating $\hat{Q}$. Importantly, this uses linear memory, as we refresh $a^+[v]$ and $b^+[v]$ after each new value in L.

So far good news, we managed to avoid quadratic memory at virtually no cost. Unfortu-

Algorithm 2 GenericJoinCRPQ-Bipartite($Q, G, \overline{x}_1$)

$A \leftarrow \emptyset$
if $|\overline{x}| = 1$ **then**
  return Eval(Q, G)
$L \leftarrow \mathsf{GenericJoin}(Q_{\overline{x}_1}, G)$
for $\mathbf{t}_{\overline{x}_1} \in L$ **do**
  for $i \in [\ell + 1, k]$ **do**
    if $y_i \in \overline{x}_1$ **then** ▷ processing $r_i(y_i, z_i)$
      $r_i[v] \leftarrow \{v' \mid (v, v') \in [\![r_i]\!]_G\}$
      Replace $r_i(y_i, z_i)$ in $\hat{Q}$ for $r_i[v](z_i)$
    else ▷ bipartite implies $z_i \in \overline{x}_1$
      $r_i[v] \leftarrow \{v' \mid (v', v) \in [\![r_i]\!]_G\}$
      Replace $r_i(y_i, z_i)$ in $\hat{Q}$ for $r_i[v](y_i)$
    $\hat{G} \leftarrow G \cup r_i[v]$
  $A[\mathbf{t}_{\overline{x}_1}] \leftarrow \mathsf{GenericJoin}(\hat{Q}, \hat{G})$
  $A \leftarrow A \cup \{\mathbf{t}_{\overline{x}_1}\} \times A[\mathbf{t}_{\overline{x}_1}]$
return A

nately, we cannot avoid it for all queries. Let us consider the triangle query but now with three RPQs: $Q(x, y, z) \leftarrow a^+(x, y) \wedge b^+(y, z) \wedge c^+(x, z)$. The cardinalities of all starting and endpoints will be N and let us assume that the first chosen variable is y so the computation goes as in the example above, except that $\hat{Q}(x, v, z) \leftarrow a^+[v](x) \wedge b^+[v](z) \wedge c^+(x, z)$ will still have one more RPQ to compute and therefore the running time will be in $O(N \cdot (N^2 + N^3))$. It is easy to see that all possible orders for this query will result in the same algorithm: for this query we cannot avoid having to nest at least the computation of two RPQs.

In the best case, thus, GenericJoinCRPQ does run in the sought after FullMaterialization time bounds. But for certain queries and orderings, the algorithm resorts to computing each RPQ on demand, which implies a much slower $O(\mathrm{AGM}(Q, G) \cdot |V|^2)$ bound.

Queries for which GenericJoinCRPQ is efficient. As we have seen, the problem in our algorithm is that nesting the evaluation of RPQs is often too costly, and sends us above the FullMaterialization bound. As it turns out, we can characterize the types of queries for which the nesting can be avoided, and introduce a version of GenericJoinCRPQ that takes advantage of this structure.

For this, we will require the query Q is such that its RPQ components form a bipartite graph. More formally, assume that we have a full CRPQ $Q(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$. We will say that Q is *RPQ-bipartite*, if the graph $G_r(Q) = (V_r, E_r)$, with $V_r = \bigcup_{i=\ell+1}^{k} \{y_i, z_i\}$, and $E_r = \{(y_i, z_i) \mid i = \ell + 1, \dots, k\}$, is bipartite. We call the graph $G_r(Q)$ the RPQ-graph of Q. Assume that Q is RPQ-bipartite, let $\overline{x}$ contain all the variables of Q, and $\overline{x}_1, \overline{x} - \overline{x}_1$ be a bipartiton of the RPQ-graph of Q. Then evaluating Q over a graph database G can be done via Algorithm 2, which generalizes GenericJoinCRPQ so that it takes the advantage of the bipartite structure of Q. Here for a CRPQ Q, and a set of variables $\overline{x}_1$, with $Q_{\overline{x}_1}$ we denote the CRPQ Q restricted to conjuncts using only the variables in $\overline{x}_1$. Notice that, given that $\overline{x}_1$ partitions the RPQ-graph of Q, the query $Q_{\overline{x}_1}$ contains only relations and no RPQs.

Algorithm GenericJoinCRPQ-Bipartite generalizes Algorithm 1 by taking the first partition of vertices to be a partition that forms a bipartition in the RPQ-graph of the query.

This allows us to instantiate the starting vertices from which all the RPQs in Q will be computed. Intuitively, the existence of a bipartition in the RPQ-graph of the query allows us to divide the query into two subqueries with no RPQs and by this avoid having to compute nested RPQs.

In order to show that the algorithm is correct and to analyse its running time, we decompose the algorithm in three parts:

1. First, we compute the tuples $\mathbf{t}_{\overline{x}_1}$ in the answer of $Q_{\overline{x}_1}$ using the relational GenericJoin (line 4).
2. For every tuple $\mathbf{t}_{\overline{x}_1}$ we compute all the associated regular expressions (lines 5–13).
3. We compute the rest of the join (involving the variables in $\overline{x} - \overline{x}_1$ with the relational GenericJoin (line 14).

In the worst case, we must perform $\mathrm{AGM}(Q_{\overline{x}_1}, G_{\overline{x}_1})$ computations of every regular expression r_i. Therefore, the total cost is in $O(\mathrm{AGM}(Q_{\overline{x}_1}, G_{\overline{x}_1}) \times |V|^2)$ (the $|V|^2$ being the cost of computing the RPQs). Next, we also need to evaluate the remaining (conjunctive) query over variables $\overline{x} - \overline{x}_1$. This takes time in $O(\mathrm{AGM}(Q_{\overline{x}-\overline{x}_1}, G_{\overline{x}-\overline{x}_1}))$. We obtain the following.

▶ **Theorem 12.** *Let $Q(\overline{x})$ be a CRPQ such that its RPQ-graph is bipartite, and let $\overline{x}'$, $\overline{x}''$ be an RPQ-bipartition, with $|\overline{x}'| \leq |\overline{x}''|$. Then the running time of* GenericJoinCRPQ-Bipartite *over Q and a graph $G = (V, E)$ is*

$$AGM(Q_{\overline{x}'}, G) \cdot |V|^2 + AGM(Q_{\overline{x}''}, G).$$

In order to reach the running time of FullMaterialization we need the query to be even further restricted. In particular, if the bipartition is such that one side contains a single variable, then the algorithm is equivalent to fixing a vertex in this variable, computing all the RPQs in Q from this vertex (by the property of bipartition, no other vertex exists), and then joining the rest using GenericJoin. This gives us the following.

▶ **Corollary 13.** *When the RPQ-graph of a CRPQ Q is bipartite and it admits a partition $\overline{x}'$, $\overline{x}''$ with $\min\{|\overline{x}'|, |\overline{x}''|\} = 1$, the running time of* GenericJoinCRPQ-Bipartite *is equal to* FullMaterialization.

Hence, for these types of CRPQs we can achieve running time of FullMaterialization using only linear memory. It is not difficult to show that GenericJoinCRPQ-Bipartite does not run under the FullMaterialization bound when queries are not of this specific shape. In general, we conjecture that this bound (under memory constraints) is not attainable when graphs are not RPQ-bipartite; solving this problem opens up an interesting line of work into space-time tradeoffs for computing the answers of a CRPQ.

5 Conclusions and future work

Our paper provides techniques for understanding size bounds of CRPQs, and makes use of these techniques to inform better algorithms for evaluating CRPQs. Our work also opens up several lines of work regarding CRPQs, size bounds and algorithms. A first important problem is to verify that GenericJoinCRPQ-Bipartite works well in practice, and enjoys as big success as standard worst-case optimal algorithms in graph databases. Of course, moving beyond RPQ-bipartite queries would require either new algorithms, or proving that the bounds offered by GenericJoinCRPQ cannot be improved. Further, there are several questions regarding tight bounds for complex classes of queries. In particular, our bounds for CRPQs with ε or RPQs accepting ε are only shown for very structured graphs where all relations share the same vertices, and it would be good to show that the bound remains to hold under arbitrary cardinalities.

References

1. Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.*, 50(5):68:1–68:40, 2017.
2. Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma, and Adrián Soto. Worst-case optimal graph joins in almost no space. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 102–114. ACM, 2021.
3. Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, and Javiel Rojas-Ledesma. Time-and space-efficient regular path queries on graphs. *arXiv preprint arXiv:2111.04556*, 2021.
4. Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013.
5. Pablo Barceló Baeza. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 175–188, 2013.
6. Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *VLDB J.*, 29(2-3):655–679, 2020.
7. Katrin Casel and Markus L. Schmid. Fine-grained complexity of regular path queries. In *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*, pages 19:1–19:20, 2021.
8. Michael J. Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, and Thomas Neumann. Adopting worst-case optimal joins in relational database systems. *Proc. VLDB Endow.*, 13(11):1891–1904, 2020.
9. Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16:1–16:35, 2012.
10. Aidan Hogan, Cristian Riveros, Carlos Rojas, and Adrián Soto. A worst-case optimal join algorithm for SPARQL. In *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I*, pages 258–275, 2019.
11. Thomas Neumann and Gerhard Weikum. Rdf-3x: a risc-style engine for rdf. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.
12. Thomas Neumann and Gerhard Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19(1):91–113, 2010.
13. Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. In *PODS 2012*, pages 37–48, 2012.
14. Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013.
15. Dung Nguyen, Molham Aref, Martin Bravenboer, George Kollias, Hung Q Ngo, Christopher Ré, and Atri Rudra. Join processing for graph patterns: An old dog with new tricks. In *Proceedings of the GRADES'15*, pages 1–8. 2015.
16. Jena Team. TDB Documentation, 2021. URL: `https://jena.apache.org/documentation/tdb/`.
17. Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 96–106. OpenProceedings.org, 2014.
18. Virginia Vassilevska Williams and R Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):1–38, 2018.
19. Peter T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, 2012.

6 Appendix

6.1 Proof of Theorem 2

The upper bound. In order to prove the upper bound we will first introduce a slightly modified version of the query decomposition lemma of [13]. Here, the query Q from our theorem is represented as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of variables of Q, and for each hyperedge $(x, y) \in \mathcal{E}$, $R_{a_i}(x, y)$ appears in Q, or $r_i(x, y)$ does, with R_{a_i} a relation, and r_i a regular expression. Naturally, we can partition $\mathcal{E} = \mathcal{E}_R \cup \mathcal{E}_r$ with $\mathcal{E}_R$ the hyperedges corresponding to relations in Q and $\mathcal{E}_r$ the hyperedges corresponding to regular expressions. For $F \in \mathcal{E}_R$, with E_F we denote the relation R_{a_i} with the variables F, and for $F \in \mathcal{E}_r$ the regular expression with the variables F. Additionally, if $J \subseteq \mathcal{V}$, by $(\mathcal{E}_R)_J$ we denote the set of all $F \in \mathcal{E}_R$ having a non empty intersection with J, and analogously for $(\mathcal{E}_r)_J$. Given this representation, a *crpq-cover* is simply a vector $\overline{u} \in \mathbb{R}^{|\mathcal{E}|}$, such that $\overline{u} \geq \mathbf{0}$, and for each $v \in \mathcal{V}$, it holds that $\sum_{F: v \in F} u_F \geq 1$.

▶ **Lemma 14** (Query decomposition lemma for full CRPQs)**.** *Let $Q(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ be a full CRPQ, represented by a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{E} = \mathcal{E}_R \cup \mathcal{E}_r$ as above. Let $\overline{u}$ be a crpq-cover for $\mathcal{H}$. Take an arbitrary partition $\mathcal{V} = I \uplus J$ such that $1 \leq |I| \leq |\mathcal{V}|$ and $L = \bigwedge_{i=1}^{\ell} \pi_I(R_{a_i}) \bigwedge_{i=\ell+1}^{k} \pi_I(r_i)$. Then*

$$
\begin{aligned}
&\sum_{t_I \in L} \prod_{F \in (\mathcal{E}_R)_J} |E_F|^{u_F} \prod_{F \in (\mathcal{E}_r)_J} |E_F^s|^{u_{y_F}^{E_F}} |E_F^e|^{u_{z_F}^{E_F}} \\
&\leq \prod_{F \in (\mathcal{E}_R)} |E_F|^{u_i} \prod_{F \in (\mathcal{E}_r)} |E_F^s|^{x_F^s} |E_F^e|^{u_F^{E^f}}
\end{aligned}
$$

Base case. For the base case, we take $|\mathcal{V}| = 2$ so the same two attributes will apear in all the RPQs. Let y and z be such attributes so without loss of generality, our query will look like,

$$
Q(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y, z) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y, z),
$$

and the proof goes as follows

$$
|Q| \leq \min\{\min_{i \in [1,\ell]} |R_{a_i}|, \min_{i \in [\ell+1,k]} |r_i^s| \cdot \min_{i \in [\ell+1,k]} |r_i^e|\}
$$

Take a crpq-cover $\overline{u}$ and let $\alpha = \sum_{i=1}^{\ell} u^{R_{a_i}}$, so $\sum_{i=\ell+1}^{k} u_{y_i}^{exp_i} \geq 1 - \alpha$ and $\sum_{i=\ell+1}^{k} u_{z_i}^{exp_i} \geq 1 - \alpha$, then

$$
\begin{aligned}
&\leq \min |R_{a_i}|^{\alpha} \cdot \min |R_{a_i}|^{1-\alpha} \\
&\leq \min |R_{a_i}|^{\alpha} \cdot \min |r_i^s|^{1-\alpha} \cdot \min |r_i^e|^{1-\alpha} \\
&\leq \min |R_{a_i}|^{\sum_{i=1}^{\ell} u^{R_{a_i}}} \cdot \min |r_i^s|^{\sum_{i=\ell+1}^{k} u_{y_i}^{r_i}} \cdot \min |r_i^e|^{\sum_{i=\ell+1}^{k} u_{z_i}^{r_i}} \\
&= \prod_{i=1}^{\ell} \min |R_{a_i}|^{u^{R_{a_i}}} \prod_{i=\ell+1}^{k} \min |r_i^s|^{u_{y_i}^{r_i}} \cdot \min |r_i^e|^{u_{z_i}^{r_i}} \\
&\leq \prod_{i=1}^{\ell} |R_{a_i}|^{u^{R_{a_i}}} \prod_{i=\ell+1}^{k} |r_i^s|^{u_{y_i}^{r_i}} \cdot |r_i^e|^{u_{z_i}^{r_i}}
\end{aligned}
$$

Inductive step. For the inductive step we will use the query decomposition lemma. First we assume that $|\mathcal{V}| \geq 3$ and choose a partition $\mathcal{V} = I \uplus J$ with $1 \leq |I| < |\mathcal{V}|$. Then we define $L = \bigwedge_{i=1}^{\ell} \pi_I(R_{a_i}) \bigwedge_{i=\ell+1}^{k} \pi_I(r_i)$ and for each tuple $t_I \in L$ we have

$$Q[t_I] \leftarrow \bigwedge_{i=1}^{\ell} \pi_J(R_{a_i} \ltimes t_I) \bigwedge_{i=\ell+1}^{k} \pi_J(r_i \ltimes t_I)$$

then the original query Q can be written as $Q = \cup_{t_I \in L}(t_I \times Q[t_I])$.

Let u be a crpq-cover for $Q[t_I]$ then the induction hypothesis gives us that

$$\begin{aligned}
|Q[t_I]| &\leq \prod_{i=1}^{\ell} |\pi_J(R_{a_i} \ltimes t_I)|^{u^{R_{a_i}}} \cdot \prod_{i=\ell+1}^{k} |\pi_J(r_i^s \ltimes t_I)|^{u_{y_i}^{r_i}} \cdot |\pi_J(r_i^e \ltimes t_I)|^{u_{z_i}^{r_i}} \\
&= \prod_{i=1}^{\ell} |R_{a_i} \ltimes t_I|^{x_i} \cdot \prod_{i=\ell+1}^{k} |r_i^s \ltimes t_I|^{x_i^s} \cdot |r_i^e \ltimes t_I|^{x_i^e}
\end{aligned}$$

then by applying the query decomposition lemma we get the desired upper bound

$$|\mathrm{Eval}(Q, \mathcal{G}))| = \sum_{t_i \in L} |Q[t_I]| \leq \prod_{i=1}^{\ell} |R_{a_i}|^{u^{R_{a_i}}} \prod_{i=\ell+1}^{k} |r_i^s|^{u_{y_i}^{r_i}} \cdot |r_i^e|^{u_{z_i}^{r_i^e}}$$

6.2 Proof of Lemma 3

Let $Q(\overline{x}) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ be a simple CRPQ, so that *flat*(Q) is the conjunctive query defined as:

$$\mathit{flat}(Q) \leftarrow \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i^s(y_i) \wedge r_i^e(z_i)$$

Consider a tuple $\bar{t}$ in $\mathrm{Eval}(Q, G)$, we need to show that it belongs to *flat*(Q). If $\bar{t}$ belongs to $\mathrm{Eval}(Q, G)$, then there is a matching μ such that for each $1 \leq i \leq \ell$ we have that $(\mu(y_i), \mu(z_i))$ belong to R_{a_i}, and for each $\ell + 1 \leq i \leq k$, $(\mu(y_i), \mu(z_i))$ belong to $[\![r_i]\!]_G$. For μ to be a matching for *flat*(Q), all we need to show is that $\mu(y_i)$ belongs to r_i^s and $\mu(z_i)$ belongs to r_i^e. Since r_i does not accept ε, we can easily show that this is a necessary condition for the fact that $(\mu(y_i), \mu(z_i)) \in [\![r_i]\!]_G$ holds.

6.3 Proof of Corollary 4

Consider the primal program, but now omiting relations and expression's endpoints cardinalities

$$\begin{aligned}
&\text{minimize} && \sum_{i=1}^{\ell} u^{R_{a_i}} + \sum_{i=\ell+1}^{k} u_{y_i}^{r_i^s} + u_{z_i}^{r_i^s} \\
&\text{where} && \sum_{i=1}^{\ell} u^{R_{a_i}} + \sum_{i=\ell+1}^{k} u_{y_i}^{r_i^s} + \sum_{i=\ell+1}^{k} u_{z_i}^{r_i^e} \geq 1 \quad \text{for } x \in \overline{x} \\
& && u^{R_{a_i}} \geq 0 \quad \text{for } i \in [1, \ell] \\
& && u_{y_i}^{r_i^s}, u_{z_i}^{r_i^e} \geq 0 \quad \text{for } i \in [\ell + 1, k]
\end{aligned} \tag{6}$$

and it's corresponding dual

$$\text{maximize: } \sum_{x\in\overline{x}} v_x$$

$$\begin{aligned}\text{subject to: } v_{y_i}+v_{z_i} &\leq 1, & i&=1,\dots,\ell\\ v_{y_i} &\leq 1, & i&=\ell+1,\dots,k\\ v_{z_i} &\leq 1, & i&=\ell+1,\dots,k\\ v_x &\geq 0, & x&\in\overline{x}\end{aligned}$$

We take an optimal solution in the primal and dual: $\overline{u}$ and $\overline{v}$ and as all the components of the program are one, the solutions must be rational. Let $v_x = p_x/q$ for every $x \in \overline{x}$ and consider an arbitrarily large $N_0 \in \mathbb{N}$. Make $N = N_0^q$. We will stablish for every attribute $x \in \overline{x}$ $\mathrm{dom}(x) = V_x$ with $|V_x| = N^{p_x/q}$ and build the instance as in the lowerbound proof of 2. With this we get that

- $|R_{a_i}| = N^{v_{y_i}+v_{z_i}} \leq N$ (with y_i and z_i the attributes of R_{a_i} and $v_{y_i}+v_{z_i} \leq 1$ by the restrictions of the dual program)
- $|exp_i^s| = N^{v_{y_i}} \leq N$
- $|exp_i^e| = N^{v_{z_i}} \leq N$

Furthermore, we have that, because of the construction, $\mathrm{Eval}(Q,G)$ has all the tuples in $V_{x_1}\times \dots V_{x_n}$ with $(x_1,\dots,x_n)=\overline{x}$, so

$$\begin{aligned}|\mathrm{Eval}(Q,G)| &= \prod_{x\in\overline{x}} N^{p_x/q} = \prod_{i=1}^{\ell}|N|^{u}\prod_{i=\ell+1}^{k}|N|^{u_{y_i}^{r_i}+u_{z_i}^{r_i}}\\ &\geq \prod_{i=1}^{\ell}|R_{a_i}|^{u^{R_{a_i}}}\prod_{i=\ell+1}^{k}|r^s|^{u_{y_i}^{r_i}}\cdot|r^e|^{u_{z_i}^{r_i}}\end{aligned}$$

To see that the cardinalities are all the same for every relation R_{a_i} with $u^{R_{a_i}} > 0$ and endpoint of expression r_i^s with $u_{y_i}^{r_i} > 0$ and r_i^e with $u_{z_i}^{r_i} > 0$ we use the conditions of complemetary slackness that gives us $\sum_{x=y_i\vee x=y_i} v_x = 1$ for $R_{a_i}(y_i,z_i)$ with $u^{R_{a_i}} > 0$, $v_{y_i} = 1$ for $r_i(y_i,z_i)$ with $u_{y_i}^{r_i} > 0$ and $v_{z_i} = 1$ for $r_i(y_i,z_i)$ with $u_{z_i}^{r_i} > 0$, so finally for those relations and endpoints we have that $|R_{a_i}| = |r_j^s| = |r_j^e| = N$.

6.4 Proof of Proposition 5

As stated in Section 3 for query Q we will define a simple query Q' in such way that

- for every repeated relations/labels we will define a fresh one
- every expression r that only accepts words on lenght 1 will be replaced by a relation R_r

Also, for every graph G compatible with Q we define a graph G' in which every fresh relation used in Q' contains exactly the same pairs the relation from which it originated in the transformation and the relations R_r will be the union of the relations corresponding to the labels of the accepted words of r. For example: if $r = a_1|a_2$ then $R_r = R_{a_i} \cup R_{a_2}$.
For the **upper bound** it suffices to see that $|\mathrm{Eval}(Q,G) = \mathrm{Eval}(Q',G')|$. As Q' is a simple CRPQ then for Theorem 2 we have $|\mathrm{Eval}(Q,G) = \mathrm{Eval}(Q',G')| \leq 2^{\rho^*(Q',G')}$.
For the **lower bound** as Q' suffices conditions of Theorem 2 (and Corollary 4), there are arbitrary graph database instances G' such that $\mathrm{Eval}(Q',G') \geq 2^{\rho^*(Q',G')}$, and, moreover we can build such instances in a way that every relations R_{a_i} and endpoints of regular expression r_i^s, r_i^e with weights in the optimal solution $u^{R_{a_i}} > 0$, $u_{y_i}^{r_i} > 0$, $u_{y_i}^{r_i} > 0$ respectively have the same cardinality. Starting from this instance G' we will create an instance G

compatible with the original query Q that meets the bound. The graph G will be just as G but with the repeated relation defined as the union of the fresh relations of G'. Certainly $|\text{Eval}(Q, G)| \geq |\text{Eval}(Q', G')| = 2^{\rho^*(Q',G')}$.

Furthermore, the linear program for Q is defined just like for simple queries, just treating regular expressions that only accepts word of lenght 1 as relations. Let $\rho^*(Q, G)$ be the optimal solution of this program over instance G defined above, then as any solution for Q' is also a valid solution for Q and all the relations and endpoints of Q that have weight greater than 0 have equal size, we have that $2^{\rho^*(Q',G')} \geq \frac{2^{\rho^*(Q,G)}}{2^{rep(Q)}}$ with $rep(Q)$ the amount of repeated relations or labels in Q.

6.5 Proof of Proposition 6

First let's present the linear program for a query $P(\overline{x}_0) \leftarrow Q(\overline{x})$ with projections: for this we will modify linear program 4 ignoring all the attributes that are not in the projections.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{\ell} u^{R_{a_i}} \ \log |R_{a_i}| \ + \sum_{i=\ell+1}^{k} (u^{r_i}_{y_i} \log |r^s_i| + u^{r_i}_{z_i} \log |r^e_i|) \\
\text{where} \quad & \sum_{\substack{i:x=y_i \vee \\ i:x=z_i}} u^{R_{a_i}} + \sum_{i:x=y_i} u^{r_i}_{y_i} + \sum_{i:x=z_i} u^{r_i}_{z_i} \geq 1 \quad \text{for } x \in \overline{x}_0 \\
& u^{R_{a_i}} \geq 0 \quad \text{for } i \in [1, \ell] \\
& u^{r_i}_{y_i}, u^{r_i}_{z_i} \geq 0 \quad \text{for } i \in [\ell+1, k]
\end{aligned}
\tag{7}
$$

Just as in the simple CRPQ case, the optimal solution to this program given an instance G will be denoted by $\rho^*_{\overline{x}_0}(Q, G)$.

We will take P and transform it into a projection of a simple and full CRPQ Q' by doing the steps of Proposition 5

- the relations/labels that are repeated will be replaced with fresh labels/relations
- the expressions r that only accepts words of lenght 1 will be replaced with relations R_r

Finally, to deal with the projections, we will take $\pi_{\overline{x}_0}(Q'(\overline{x}))$. From this we define a simple CRPQ Q'' just like $\pi_{\overline{x}_0}(Q'(\overline{x}))$ but ignoring all the attrbutes that are projected out.

Also, for every instance G compatible with P, a graph G'' compatible with Q'' such that $|\text{Eval}(P, G)| \leq |\text{Eval}(Q'', G'')|$ by

- projecting the attributes of relations and expressions
- every fresh relation used in Q' contains exactly the same pairs the relation from which it originated in the transformation
- relations R_r will be the union of the relations corresponding to the labels of the accepted words of r

For the **upper bound**, as the modified query Q'' is a simple CRPQ, we have $|\text{Eval}(Q'', G'')| \leq 2^{\rho^*(Q'',G'')}$. Also, $|\text{Eval}(P, G)| \leq |\text{Eval}(Q'', G'')| \leq |\text{Eval}(Q'', G'')|$ (with G' as G'' but not projecting the attributes). Moreover, as the linear program for both $\pi_{\overline{x}_0}(Q'(\overline{x}))$ and Q'' differ only on the cost function (same restrictions), we have that $\rho^*(Q'', G'') \leq \rho^*_{\overline{x}_0}(Q', G')$ and therefore $|\text{Eval}(P, G)| =\leq 2^{\rho^*_{\overline{x}_0}(Q',G')}$.

For the **lower bound**, we see first that as Q'' is a simple CRPQ, then there's a graph instance G'' such that $|\text{Eval}(Q'', G'')| \geq 2^{\rho(Q'',G'')}$. From G' we will create an instance for P in a similar fashion that we did for full CRPQs in Proposition 5:

- repeated relations/labels are defined as the union of the fresh relations in G''

- in the spots of every attribute x that is projected out in P we will extend relations and expressions so $\pi_x(R_{a_i}) = \{\star\}$ and $\pi_x(r_i) = \{\star\}$ with $\star$ an arbitrary node of the graph.

Combining this construction with the result from Proposition 5 we have that $|\text{Eval}(P, G)| = |\text{Eval}(Q'', G'')| \geq 2^{\rho^*(Q'',G'')} \geq \frac{2^{\rho^*_{\bar{x}_0}(Q',G')}}{2^{rep(P)}}$

6.6 Proof of Lemma 7

Immediate form the fact that both $\varepsilon(x, y)$ and $V(x) \wedge V(y) \wedge x = y$ always produce D as their answers.

6.7 Proof of Lemma 8

The reason why Q and chase(Q) are not equivalent is because the chase process may incur in the loss of a variable in the output answer of Q. To see that $|\text{Eval}(Q, G)| \geq |\text{Eval}(\text{chase}(Q), G)|$, let $y_1, \ldots, y_p$ be the set of variables that was replaced in the chase process. We assume that each $y_1, \ldots, y_p$ was replaced for a variable $x_1, \ldots, x_p$ which is actually used in chase(Q) (if not, consider that all y_i's that are replaced by the same variable belong to an equivalence class, and choose one representative of each such class). Now notice that every mapping μ from chase(Q) to G can be augmented into a mapping μ' for Q over G by letting $\mu'(x_p) = \mu(y_p)$. It must then be the case that every tuple in Eval(chase$(Q), G$) can be assigned its augmented tuple in Eval(Q, G).

$|\text{Eval}(Q, G)| \leq |\text{Eval}(\text{chase}(Q), G)|$, we again focus on all of $y_1, \ldots, y_p$ and $x_1, \ldots, x_p$ in Q. From the definition of the chase, there must be equalities $y_i = x_i$ in Q, for $1 \leq i \leq p$. This means that the cardinality of Eval(Q, G) is the same as the cardinality of the set Eval(Q, G) when positions corresponding to $y_1, \ldots, y_p$ are projected out. Further, to every mapping μ from Q to G, the corresponding mapping μ' from Q to G, the same as μ except $y_1, \ldots, y_p$ is not in the pre-image. By the definition of chase(Q), μ' is a valid mapping from chase(Q) to G. All in all, this means that every tuple in the set Eval(Q, G) when positions corresponding to $y_1, \ldots, y_p$ are projected out, is an answer to chase(Q).

6.8 Proof of Proposition 9

Follows immediately from 5 and Lemmas 7 and 8: if the evaluation of chase$(P^{\setminus\varepsilon})$ over a graph G is bounded by a number, then so is P. On the other hand, if there are arbitrarily large graphs for which the evaluation of chase$(P^{\setminus\varepsilon})$ over a graph G is bigger than an expression, then the evaluation of P over this graph is also bound to be bigger than this expression.

6.9 Proof of Propostion 10

First of all, let's introduce a generalized version of Corollary 4 for ε-free non-full CRPQs:

▶ **Lemma 15.** *Given an ε-free CRPQ P, we can build an arbitrarily large instance G such that $|Eval(P, G)| 2^{p(|Q|)} \geq 2^{\rho^*_{\bar{x}_0}(Q',G')}$ with Q' and G' defined as in Section 3.3, $|R_{a_i}| = |r^s_j| = |r^e_j|$ for every relation i and j such that $u^{R_{a_i}} > 0$, $u^{r_j}_{y_j} > 0$ and $u^{r_j}_{z_j} > 0$.*

Proof. This lemma is proved using the same techniques from the lower bound proof of Proposition 6. We start with an ε-free query Q and transform it into a simple CRPQ Q'', by creating new relations out of repeated labels/relations and expressions that only accepts words of lenght 1 and then removing the projected attributes. Then, by Corollary 4 there's an instance G'' such that $|\text{Eval}(Q'', G''))| \geq 2^{\rho^*(Q'',G'')}$ with same cardinalities in all relations and endpoints with weight greater than 0. From this instance G'', we can create an instance G compatible with Q where again all the relations and endpoints with weight greater than 0 have the same cardinality: in spots of projected attributes, we extend relations so $\pi_x(R_{a_i}) = \{\star\}$ which dosen't change the cardinality of relations and we define repeated relations as the union of the fresh relations in G'' (which actually have the same tuples). As seen in Proposition 15, $|\text{Eval}(P, G)| \cdot 2^{rep(P)} \geq 2^{\rho^*_{\bar{x}_0}(Q',G')}$. ◀

For the **lower bound**, we proceed in a similar fashion to what we did for Proposition 9. Consider query $Q[S]$ where $S = \{\ell_1, \ldots, \ell_p\}$, which is ε-free, and invoke the lower bound of Lemma 15, in which all relations and vertices with positive weight in the program have equal cardinality, say N. From this graph G, we create a second graph G' in which:

- We replace all values corresponding to every vertex of the query with $\{1, \ldots, N\}$
- Replace element $\star$ with element 1

Since going from G to G' we are equating elements and adding more relations, it is clear that $|\text{Eval}(Q[S], G)| \leq |\text{Eval}(Q[S], G')|$, as any match for Q over G can be transformed to a match for Q over G' be equating relevant elements as we did for G'. Moreover, the linear program does not change for G', as we continue to impose that all cardinalities are at most N. Finally, note that in G' we have that $[\![r]\!]_{G'} = \{1, \ldots, N\} \times \{1, \ldots, N\}$, which means that $[\![\varepsilon]\!]_{G'} \subseteq [\![r]\!]_{G'}$ and all relations K_ℓ in G' are empty. It follows that $|\text{Eval}(Q[S'], G')| = 0$ for any $S' \neq \{\ell_1, \ldots, \ell_p\}$, and therefore $|\text{Eval}(Q, G')| = |\text{Eval}(Q[S], G')| > 2^{\rho^*(Q,G')}$, which was to be shown.

For the **upper bound**. Let $\bar{t}$ be a tuple in $\text{Eval}(Q, G)$. We show that there is at least one set S such that $\bar{t}$ is in $\text{Eval}(\mathit{flat}(Q[S]), G)$. To construct such set S, for each atom $r_{l_j}(y_{l_j}, z_{l_j})$ in Q, where r_{l_j} can be decomposed in $\varepsilon + \hat{r}_{l_j}$, with the latter an ε-free expression. Consider the values t and t' in positions corresponding to variables y_{l_j} and z_{l_j}, respectively. Now, if $t \in \hat{r}^s_{l_j}$ and $t' \in \hat{r}^e_{l_j}$, then add ℓ_j to S, otherwise continue to the next atom.

With S defined, we show how to extend the match μ witnessing $\bar{t} \in \text{Eval}(Q, G)$ is also a match for $\text{Eval}(\mathit{flat}(Q[S]), G)$. This follows immediately from Lemma 3 for any atom not of the form $r_\ell(y_\ell, z_\ell)$ for $\ell \in \{\ell_1, \ldots, \ell_p\}$. For atoms of the form $r_\ell(y_\ell, z_\ell)$ for $\ell \in \{\ell_1, \ldots, \ell_p\}$, we distinguish two cases.

- $t \in \hat{r}^s_{l_j}$ and $t' \in \hat{r}^e_{l_j}$. Then, $l_j \in S$. Further, μ can be extended to satisfy atoms $\hat{r}^s_{l_j}(y_{l_j}) \wedge \hat{r}^e_{l_j}(z_{l_j})$ in $\mathit{flat}(Q[S])$.
- $t \notin \hat{r}^s_{l_j}$ or $t' \notin \hat{r}^e_{l_j}$. Then by definition, the pair (t, t') does not belong to $[\![\hat{r}_{l_j}]\!]_G$. Thus, it must be that $t = t'$, but l_j is not in S. This means both that $\mathit{flat}(Q[S])$ contains, instead of r_l, atoms $K_{\ell_j}(t) \wedge K_{\ell_j}(t') \wedge t = t'$, and also that G agrees with those atoms, which implies that μ can be extended to $\mathit{flat}(Q[S])$.

One important caveat is that so far we know how to bound queries using arbitrary equalities over unary relations, apart from relation V. However, these expressions can be *chased* just as ε, and for them we can show an analog of Proposition 9.

6.10 Proof of Theorem 12

Following the decomposition of Algorithm 2, we see that the first to be done is to compute L on line 4. Because of the bipartition condition, this is done via the generic join for CQs and therefore the cost is in $O(\text{AGM}(Q_{\overline{x}'}, G))$. Then for every tuple in L, we compute each of the RPQs of Q. The cost to compute a simple RPQ is in $O(|V|^2)$. In total, the cost of computing the RPQs is in $O(\text{AGM}(Q_{\overline{x}'}, G)) \cdot |V|^2)$. After computing and updating the graph, we know that the second call to the standard, CQ-based generic join is in $O(\text{AGM}(Q_{\overline{x}''}, G''))$.

However, since $G'' \subseteq G$, we also have that the cost of computing the join of the remaining part of the query is in $O(\mathrm{AGM}(Q_{\overline{x}''}, G))$. Summing up, the running time of Algorithm 2 is $O(\mathrm{AGM}(Q_{\overline{x}'}, G) \cdot |V(G)|^2 + \mathrm{AGM}(Q_{\overline{x}''}, G))$.