PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SIZE BOUNDS AND ALGORITHMS FOR CONJUNCTIVE REGULAR PATH QUERIES

TAMARA A. CUCUMIDES FAUNDEZ

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
DOMAGOJ VRGOC, PH.D

Santiago de Chile, Enero 2022

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

SIZE BOUNDS AND ALGORITHMS FOR CONJUNCTIVE REGULAR PATH QUERIES

TAMARA A. CUCUMIDES FAUNDEZ

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, Enero 2022

With all my love to my family,

boyfriend and cats.

ACKNOWLEDGEMENTS

First of all I want to express my deep sense of gratitude to Domagoj and Juan for being amazing both in an academic and human sense. I could not have done this work without your support.

I want to thank my family: Juan Emilio, Karen, Francisca and Katalina for all their support during my whole life. Thanks for supporting me to pursue the things I wanted. An special mention to my little sub-family: my partner Johnny and our furry roommates Mozarella, Bebelin and Tokita: you guys make our house my favourite place in the world

To my life-friends: Choco, Efe and Yoda, you guys are the greatest friends I could have asked for. Also a very special thanks to Adrian who encouraged me to pursue this path and was a most appreciated confidant during all this time.

I also want to thanks Cristian Riveros, Mauricio Correa and Pablo Barceló for inviting me to work with them as a teaching assistant several times: that marked my time in college. Here the special mention goes to Cristian, who was the one that introduced me to the topics covered in this work.

To finish, I want to cite Snoop Dogg: *Last but not least, I wanna thank me I wanna thank me for believing in me, I wanna thank me for doing all this hard work.*

This work was supported by the IMFD (Instituto Milenio Fundamentos de los Datos).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

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, as these depend much more heavily on the structure of queries. 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.

Keywords: graph databases, regular path queries, worst-case optimal algorithms.

RESUMEN

Las conjuctive regular path queries (CRPQs) son una de las principales clases de consultas que se realizan sobre bases de datos de grafos. Su estructura se deriva de la estructura de consultas relacionales de joins, pero además permiten caminos de largo arbitrario para conectar puntos que deben ser considerados en dichos joins. Sin embargo y pese a su popularidad, hasta este momento se conoce poco acerca de cuáles son los mejores algoritmos para procesar CRPQs. En este trabajo nos enfocamos en algoritmos óptimos en el peor caso, esto es, algoritmos cuyo tiempo de ejecución esta acotado por la cota superior del tamaño de las consultas que procesan. Aplicaciones recientes de este tipo de algoritmos se han llevado a cabo para consultas simples en grafos con resultados promisorios. En esta tesis, mostramos que la famosa cota sobre el número de resultados de una consulta propuesta por Atserias, Grohe y Marx puede ser extendida para CRPQs, pero que para obtener cotas ajustadas, se debe trabajar con un perfil de cardinalidad ligeramente mayor. Además de establecer dicha cota, discutimos acerca de los algoritmos que provienen de esta: si se procesa y materializa previamente todas las consultas del grafo (lo que demanda memoria cuadrática en términos de los vértices del grafo), entonces las técnicas desarrolladas para consultas conjuntivas pueden ser aplicadas. Por otro lado, si se impone una restricción en la memoria de trabajo de los algoritmos, entonces estos deben ser adaptados con cuidado: el orden de variables con el cual se procesa las consultas puede tener un gran impacto sobre su tiempo de ejecución.

Palabras Claves: bases de datos de grafos, consultas de caminos, algoritmos óptimos en el peor caso.

1. INTRODUCTION

Graph patterns constitute the basic building block of most query languages for graph databases, and algorithms that can process the answers of graph patterns are important in the world. 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 (Team, 2021; Neumann & Weikum, 2008, 2010), or by implementing worst-case optimal algorithms over graphs, which run in time given by the AGM bound of queries (Nguyen et al., 2015; Hogan, Riveros, Rojas, & Soto, 2019; Arroyuelo, Hogan, Navarro, Reutter, et al., 2021), or even with a mix of both approaches (Freitag, Bandle, Schmidt, Kemper, & Neumann, 2020).

However, the majority of this work focuses on the simplest case where patterns are simply small graphs of interest, or *conjunctive queries* (CQs) that one requires to match in the complete 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 (Angles et al., 2017), but so far we know little about algorithms that can process these queries.

Consider for instance the CRPQ $Q_1(x, y, z) = a^+(x, y) \wedge R_b(y, z) \wedge R_c(x, z)$. 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. The query Q_1 then 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 standard graph pattern or conjunctive query over these materialized relations, whose answers we already know how to compute (Veldhuizen, 2014; Ngo, Ré, & Rudra, 2013). In our case, this means computing the transitive closure R_a^+ of R_a, as a virtual relation,

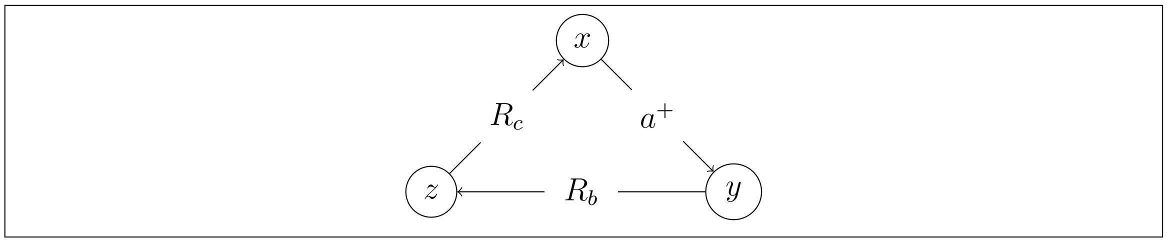

Figure 1.1. Representation of query $Q_1(x, y, z) = a^+(x, y) \wedge R_b(y, z) \wedge R_c(x, z)$. Both R_b and R_c represent standard relations and a^+ is a regular expression.

and then go on to process 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 all of R_a, R_b and R_c is N. If we use a worst-case algorithm for the task of processing 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.

On the other hand, our strategy of materializing transitive closures (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 thesis we provide answers to these questions. We study bounds on the maximum number of tuples a CRPQ may produce, given certain cardinality information about the graph. We then use these bounds to investigate optimal algorithms for CRPQs, either in full generality, or with additional memory constraints.

To be more precise, this thesis provides the following contributions.

(i). 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**. Consider the following pattern for the query Q_2:

$$\begin{tikzcd} x \arrow[r, "a^+"] & y \arrow[r, "b^+"] & z \end{tikzcd}$$

Figure 1.2. Representation of $Q_2(x, y, z) = a^+(x, y) \wedge b^+(y, z)$: a join between two regular expressions a^+ and b^+.

If $|R_a| = |R_b| = N$ then each of R_a^+ and R_b^+ may have up to N^2 tuples. Thus, applying the usual AGM bound over the CQ resulting from materializing both R_a^+ and R_b^+ 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 different tuples that we could produce is $O(N^3)$. One can show that this bound is actually tight.

(ii). 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. To be more precise, let us assume that G contains M nodes, that is, 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**. It should be noted that the cardinality of each component in a graph relation is a standard profile in graph systems using relations as their underlying storage mechanism (see e.g. (Team, 2021)). Remarkably, our bounds are still meaningful when we just assume that the projection of R_a over any of its first or second component is bounded by $|R_a|$. Furthermore, these bounds are tight (up to a factor depending on the query), and we can even get rid of this factor for patterns that do not repeat labels in edges or in path queries.

Our upper bound is based on an extension of the traditional linear program used to show the AGM bound. Consider for example query $Q_3(x, y, z) = a^+(x, y) \wedge b^+(y, z) \wedge R_c(x, z)$ in Figure 1.3.

$$\begin{tikzcd}
& x \arrow[dl, "R_c"'] \arrow[dr, "a^+"] & \\
z & & y \arrow[ll, "b^+"]
\end{tikzcd}$$

Figure 1.3. Representation of query $Q_3(x, y, z) = a^+(x, y) \wedge b^+(y, z) \wedge R_c(x, z)$ consisting of two regular expressions: a^+ and b^+ and a standard relation R_c.

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 we show that 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.

$$\begin{aligned}
\text{minimize} \quad & u^{R_c} \log|R_c| + u_x^{a^+} \log|R_a^s| + u_y^{a^+} \log|R_a^e| + \\
& u_x^{b^+}|R_b^s| + u_y^{b^+}|R_b^e| \\
\text{where} \quad & u^{R_c} + u_x^{a^+} \geq 1 \\
& u^{R_c} + u_z^{b^+} \geq 1 \\
& u_y^{a^+} + u_y^{b^+} \geq 1 \\
& u^{R_c}, u_x^{a^+}, u_y^{a^+}, u_y^{b^+}, u_z^{b^+} \geq 0
\end{aligned} \tag{1.1}$$

This is a generalization of the AGM linear program (Atserias, Grohe, & Marx, 2013), 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 (a, b) in R_c there are at most M nodes connected to a and b by means of the expressions a^+ and b^+.

(iii). 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 a graph G, the answers of Q over G are processed in time bounded by the maximum number of tuples in the answer of Q over any graph with the same cardinalities of G. Unfortunately we show that, under usual complexity assumptions, **there are CRPQs for which no worst-case optimal exists.**

(iv). 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 (Veldhuizen, 2014). For some queries, such as the triangle query in Figure 1.3, 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.

The rest of this thesis is organized ad follows: In Chapter 2 we introduce some background in both graph databases and results on bounds and worst-case optimal algorithms for traditional relational queries. Chapters 3 and 4 contain the major results of this work. In particular, Chapter 3 contain the theorems related with the bounds on the size of the answers of a CRPQ and Chapter 4 presents both a proof that worst-case optimal algorithms for any CRPQ don't exists along with an efficient algorithm to solve CRPQs. Finally, Chapter 5 provides concluding remarks and future lines of work.

2. BACKGROUND

2.1. Graphs and queries

A *graph database* is usually defined in the theoretical literature as a directed edge-labelled graph (Baeza, 2013; Wood, 2012). 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 $\{R_a(S, E)\}_{a\in\Sigma}$ 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'. In order to reuse known results for relational joins, we will often switch between graphs and their relational representations.

For a binary relation $R_a(S, E)$, with $a \in \Sigma$, we denote with R_a^s the projection of R_a onto its first attribute; namely, $R_a^s(v)$ holds if and only if there exists v' such that $R_a(v, v')$. Similarly we define R_a^e, the projection onto the second attribute. In order to reason about bounds on graph databases, we always assume the following cardinalities are also available for each $a \in \Sigma$:

- $|V|$ the total number of nodes;
- $|R_a|$, the number of a-labelled edges;
- $|R_a^s|$, the number of starting vertices of a-labelled edges;
- $|R_a^e|$, the number of end vertices of a-labelled edges.

These statistics are rather natural to consider, and are part of many graph database systems that use relations as their underlying storage mechanism (Team, 2021).

Table 2.1. 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.

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

2.2. 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 2.1 for the definition. We always assume RPQs are given both by regular expressions or automata, and freely switch between these representations.

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. Formally, a conjunctive regular path query over a graph database is given by an expression

$$Q(\overline{x}) :- \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i) \tag{2.1}$$

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 $\overline{x} = \{x_1, \ldots, x_n\} \subseteq \{y_1, z1, \ldots, y_k, z_k\}$ is a set of output variables.

The semantics of a CRPQ Q, over a graph G is given via homomorphisms (Angles et al., 2017). 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 Eval(Q, G). A CRPQ Q is compatible with a graph G if the graph features all relations mentioned in Q.

2.3. AGM bound and worst-case optimal algorithms

Here we summarize the main components of the AGM bound (Atserias et al., 2013) and worst case optimal algorithms (Veldhuizen, 2014; Ngo et al., 2013) that follow from it.

The AGM bound. Atserias, Grohe and Marx (Atserias et al., 2013) link the size bound of a relational join query to the optimal solution to a given linear program. In graph terms[1], let $Q(x_1, \ldots, x_n) = \bigwedge R_{a_i}(y_i, z_i)$ be a join query, in which $a_i \neq a_j$, and let G be a graph database where the size of each R_{a_i} is N_i. Atserias et. al. (Atserias et al., 2013) show that an optimal bound is achieved by considering the following linear program:

$$\begin{aligned} &\text{minimize} && \sum_i x_i \, logN_i \\ &\text{where} && \sum_{\text{i : } w \text{ appears in atom } R_i} x_i \geq 1 \quad \text{for each variable } w \text{ in } Q \\ & && x_i \geq 0 \qquad\qquad\qquad\qquad \text{for } i = 1, \ldots, m \end{aligned} \tag{2.2}$$

For any solution $\mathbf{x} = (x_1, \ldots, x_m)$ of this linear program, we have:

$$|\text{Eval}(Q, D)| \leq \prod_{i=1}^{m} N_i^{x_i} = 2^{\sum_i x_i logN_i}.$$

We denote by $\rho^*(Q, D)$ the optimal value of $\sum_i x_i \, logN_i$. The AGM bound (Atserias et al., 2013) can then be stated as follows.

Theorem 2.1 (AGM bound)**.** *Let Q be a full join query using relations $R_1, \ldots, R_m$. If D is a database instance with $|R_i| = N_i$, and $\rho^*(Q, D)$ the optimal solution of the*

[1]The AGM bound for conjuctive queries works with relations of any arity. As we're working with graph databases, the relations we work with are binary.

associated linear program (2.2). Then it holds that

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

Furthermore, if $R_1, \ldots, R_m$ are all distinct, then there are arbitrary large instances D for which we have $|\mathit{Eval}(Q,D)| = 2^{\rho*(Q,D)}$.

Alternatively, we can represent the query Q as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. Here $\mathcal{V} = \{A_1, \ldots, A_n\}$ is the set of attributes appearing in Q, and $\mathcal{E}$ contains all the hyperedges F such that F is precisely the attribute set of some relation R_i appearing in Q. For instance, if $Q = R(A,B,C) \bowtie S(B,D)$, then $\mathcal{V} = \{A,B,C,D\}$, and $\mathcal{E} = \{\{A,B,C\},\{B,D\}\}$. Recall that a hyperedge is simply a set of nodes (of arbitrary size) of the hypergraph. Notice that with this representation we can write our query Q as $\bowtie_{F\in\mathcal{E}} R_F$, where R_F is simply the relation $R_i(F)$.

Worst-case optimal algorithms. Besides giving a tight bound to relational join queries, the AGM bound also gives rise to several algorithms that can guarantee that they run in the number of steps that is proportional to the worst possible output size, as guaranteed by Theorem 2.1. Algorithm 1 presents the Generic-Join from (Ngo et al., 2013). From a theoretical perspective this is the easiest worst-case optimal algorithm to analyse, as it is basically a recursive algorithm that picks a partition of the query variables and recursively compute the join, by evaluating the portion of the query that amounts to projecting on one set of variables, and then joining this result with the projection to remaining variables.

Both the AGM bound for conjuctive queries and the worst-case optimal algorithms inspired by it are the main basis of our work as we will take advantage of these results by extending them to CRPQs on graph databases.

Algorithm 1 Generic-Join($\Join_{F\in\mathcal{E}} R_F$)

```
1: $Q \leftarrow \emptyset$
2: if $|\mathcal{V}| = 1$ then
3:     return $\bigcap_{F\in\mathcal{E}} R_F$
4: Pick a partition $\mathcal{V} = I \cup J$ such that $1 \leq |I| < |\mathcal{V}|$
5: $L \leftarrow$ Generic-Join($\Join_{F\in\mathcal{E}_I} \pi_I(R_F)$)
6: for $\mathbf{t}_I \in L$ do
7:     $Q[\mathbf{t}_I] \leftarrow$ Generic-Join($\Join_{F\in\mathcal{E}_J} \pi_J(R_F \Join \mathbf{t}_I)$)
8:     $Q \leftarrow Q \cup Q[\mathbf{t}_I] \times \{\mathbf{t}_I\}$
9: return $Q$
```

3. SIZE BOUNDS ON 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^s_a with all elements in R^e_a, thus invoking a quadratic jump in terms of the size of the potential vertices matching to x and to y. For this reason, in order to provide tight output bounds, we need to extend the cardinality profile of graphs, and consider the number of starting points and ending points of any relation representing a label mentioned in some regular expression. Together with this new cardinality profile, the quadratic extension by RPQs gives rise to a modified linear program, extending that of (Atserias et al., 2013), that we use to provide our size bounds.

3.1. Motivation: underlying flat CQs

To see the intuition for our linear program, let us recall query $Q_3(x, y, z)$ in Figure 1.3, and consider a graph G.

In order to bound the size of Eval(Q_3, G), we reason in terms of the size of $[a^+]_G$. In the worst possible case, we have that $[a^+]_G = R^s_a \times R^e_a$, that is, any node from R^e_a can be reached from any node from R^s_a. It is then easy to see that the answers in the evaluation Eval(Q_3, G) will always be contained in what we call the *flat* conjunctive query

$$\mathit{flat}(Q_3)(x, y, z) = R^s_a(x) \wedge R^e_a(y) \wedge R^s_b(y) \wedge R^e_b(z) \wedge R_c(x, z),$$

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^s_a, R^e_a, R^s_b and R^e_b are unary relations in G, we have that $|\text{Eval}(Q, G)| \leq |\text{Eval}(\mathit{flat}(Q_3), G)|$, and this hold for any graph G compatible with Q. Now $\mathit{flat}(Q_3)$ is a full join query, and we know how to bound its output (Atserias et al., 2013), which immediately results in an upper bound for Q_3.

Interestingly, the focus on flat conjunctive queries has another intuitive reading. Consider again query Q_2 from Figure 1.2, its flat version is simply a cross product of unary relations

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

For a graph G in which all of R_a^s, R_a^e, R_b^s and R_b^e have exactly 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 join queries (Atserias et al., 2013) or conjunctive queries (Gottlob, Lee, Valiant, & Valiant, 2012). 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 CRPQs without projection, 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 recover the general upper and lower bounds exactly as they where stated by Atserias et al. in their seminal paper(Atserias et al., 2013). 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 (Bonifati, Martens, & Timm, 2020). The expression ε, which returns the "diagonal" $\{(v, v) : v \in V\}$ in a graph, poses hard problems at the time of bounding outputs. As we see, we can deal with it separately by using the idea of color codings introduced by Gottlob et al. (Gottlob et al., 2012), but the general case requires a bit more work.

3.2. Simple CRPQs

Let r be a regular expression. 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 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. These cardinalities will be a crucial addition to the linear program of (Atserias et al., 2013), that will allow us to reason about the output size of a CRPQ. 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.

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 $Q(\overline{x}) = \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ 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;
- The language of each r_i contains 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'.

We are ready to give a tight bound for simple CRPQs. 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. More precisely, we have the following

Theorem 3.1 (Bound for simple CRPQs)**.** *Assume that the query*

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

is a simple CRPQ. Then for any graph G we have that

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

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

$$\begin{aligned}
&\textit{minimize} && \sum_{i=1}^{\ell} u^{R_{a_i}} \ \log|R_{a_i}| \ + \sum_{i=\ell+1}^{k} u^{r_i}_{y_i} \log|r^s_i| + u^{r_i}_{z_i} \log|r^e_i| \\
&\textit{where} && \sum_{\substack{i:x=y_i\\ i:x=z_i}} u^{R_{a_i}} + \sum_{i:x=y_i} u^{r_i}_{y_i} + \sum_{i:x=z_i} u^{r_i}_{z_i} \geq 1 \quad \textit{for } x \in \overline{x} \\
& && u^{R_{a_i}} \geq 0 \quad \textit{for } i \in [1, \ell] \\
& && u^{r_i}_{y_i}, u^{r_i}_{z_i} \geq 0 \quad \textit{for } i \in [\ell + 1, k]
\end{aligned} \tag{3.1}$$

Furthermore there are arbitrarily large instances for which

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

The upper bound. We can obtain a simple upper bound proof by using flat CQs. Let $Q(\overline{x})$ be a simple CRPQ. Its underlying *flat* query *flat*(Q) is the conjunctive query defined as:

$$\mathit{flat}(Q) = \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r^s_i(y_i) \wedge r^e_i(z_i) \tag{3.2}$$

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 3.1. *$\mathit{Eval}(Q, G) \subseteq \mathit{Eval}(\mathit{flat}(Q), G)$, with Q a simple CRPQ.*

PROOF. Let $Q(\overline{x}) = \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ be a simple CRPQ, so that *flat*(Q) is the conjunctive query defined in 3.2.

Consider a tuple $\overline{t}$ in Eval(Q, G), we need to show that it belongs to *flat*(Q). If $\overline{t}$ belongs to 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. □

Since the linear programs of both *flat*(Q) (as in (Atserias et al., 2013)) and Q (as in the statement of Theorem 3.1) coincide, and $2^{\rho^*(\mathit{flat}(Q),G)}$ is an upper bound for Eval(*flat*$(Q), G)$, this immediately proves the upper bound of Theorem 3.1.

Alternatively, we can prove the upper bound directly as done in (Ngo et al., 2013). For this, we will first introduce a slightly modified version of the query decomposition lemma of (Ngo, Porat, Ré, & Rudra, 2012). 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 3.2 (Query decomposition lemma for full CRPQs)**.** *Let*

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

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

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

PROOF OF THEOREM 3.1, UPPER BOUND. The proof goes by induction over the size of $\mathcal{V}$.

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,

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

and we begin the proof stating that

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

Now we take a solution to the linear program 3.1 $\overline{u}$ and let $\alpha = \sum_{i=1}^{\ell} u^{R_{a_i}}$. As the sum for each vertex must be equal or grater than 1, we have that $\sum_{i=\ell+1}^{k} u_{y_i}^{exp_i} \geq 1 - \alpha$ and $\sum_{i=\ell+1}^{k} u_{z_i}^{exp_i} \geq 1 - \alpha$, then

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

$$= \prod_{i=1}^{\ell} \min |R_{a_i}|^{u^{R_{a_i}}} \prod_{i=\ell+1}^{k} \min |r_i^s|^{u_{y_i}^{r_i}} \cdot \min |r_i^e|^{u_{z_i}^{r_i}}$$

$$\leq \prod_{i=1}^{\ell} |R_{a_i}|^{u^{R_{a_i}}} \prod_{i=\ell+1}^{k} |r_i^s|^{u_{y_i}^{r_i}} \cdot |r_i^e|^{u_{z_i}^{r_i}}$$

Inductive step. For the inductive step we will use Lemma 3.2. 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

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

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

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

$$= \prod_{i=1}^{\ell} |R_{a_i} \ltimes t_I|^{x_i} \cdot \prod_{i=\ell+1}^{k} |r_i^s \ltimes t_I|^{x_i^s} \cdot |r_i^e \ltimes t_I|^{x_i^e}$$

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

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

□

The lower bound. As usual, the lower bound is shown 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 $Q_3(x,y,z) = a^+(x,y), b^+(y,z), R_c(x,z)$ and lets construct the linear program for this query as stated in Theorem 3.1.

$$\begin{aligned}
\text{minimize} \quad & u^{R_c} \log|R_c| + u_y^{b^+} \log|(b^+)^s| + u_z^{b^+} \log|(b^+)^e| \\
& + u_x^{a^+}|(a^+)^s| + u_y^{a^+}|(a^+)^e| \\
\text{where} \quad & u^{R_c} + u_x^{a^+} \geq 1 \\
& u^{R_c} + u_z^{b^+} \geq 1 \\
& u_y^{a^+} + u_y^{a^+} \geq 1 \\
& u^{R_c}, u_y^{b^+}, u_z^{b^+}, u_x^{a^+}, u_y^{a^+} \geq 0
\end{aligned} \tag{3.3}$$

The linear program 3.2 has the following corresponding dual program,

$$\begin{aligned}
\text{maximize: } & v_x + v_y + v_z \\
\text{subject to: } & v_x + v_z \leq \log|R_c| \\
& v_x \leq \log|(a^+)^s| \qquad v_y \leq \log|(a^+)^e| \\
& v_y \leq \log|(b^+)^s| \qquad v_z \leq \log|(b^+)^e| \\
& v_x, v_y, v_z \geq 0
\end{aligned}$$

Consider an optimal solution for the primal $\overline{u}$ 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 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, y) for every pair of nodes $(x,y) \in V_x \times V_z$
- 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 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're ready to formalize the construction above for an arbitrary CRPQ.

PROOF OF THEOREM 3.1, LOWER BOUND. As before, we use the dual program of equation (3.1)

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

Consider an instance with cardinalities $|R_{a_i}| = N_i$ for $i \in [1, \ell]$, $|r^s_j| = N^s_j$ and $|r^e_j| = N^e_j$ for $j \in [\ell+1, k]$. By duality we have that for any solution $\overline{u}$ to the primal and $\overline{v}$ for the dual, it is the case that

$$\sum_{i=1}^{\ell} u^{R_{a_i}} \log R_{a_i} + \sum_{i=\ell+1}^{k} u^{r_i}_{y_i} \log|r^s_i| + u^{r_i}_{z_i} \log|r^e_i| \geq \sum_{x\in\overline{x}} v_j,$$

with equality when the solutions are optimal. Let us assume that all N_i, N^s_i and N^e_i 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^q_i. Now we present a graph database G with $|R_i(D)| = N^q_i$, $|r^s_i| = (N^s_i)^q$ and $|r^e_i| = (N^e_i)^q$ such that $|\text{Eval}(Q, \mathcal{G})| \geq 2^{\rho^*(Q,\mathcal{D})}$.

- 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 path $\pi_i = a_{i_1} \dots a_{i_N}$ 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:

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

Further, we also verify that Eval(Q, G) contains all tuples $t \in V_{x_1} \times \dots \times V_{x_n}$. 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:

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

□

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

Corollary 3.1. *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 $u^{R_{a_i}} > 0$, $u_{y_j}^{exp_j} > 0$ and $u_{z_j}^{exp_j} > 0$.*

PROOF. Consider the primal program, but now omiting relations and expression's end-points cardinalities

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{\ell} u^{R_{a_i}} + \sum_{i=\ell+1}^{k} u_{y_i}^{r_i^s} + u_{z_i}^{r_i^s} \\
\text{where} \quad & \sum_{i=1}^{\ell} u^{R_{a_i}} + \sum_{i=\ell+1}^{k} u_{y_i}^{r_i^s} + \sum_{i=\ell+1}^{k} u_{z_i}^{r_i^e} \geq 1 \quad \text{for } x \in \overline{x} \\
& u^{R_{a_i}} \geq 0 \quad \text{for } i \in [1, \ell] \\
& u_{y_i}^{r_i^s}, u_{z_i}^{r_i^e} \geq 0 \quad \text{for } i \in [\ell+1, k]
\end{aligned} \tag{3.4}$$

and it's corresponding dual

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

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 $x \in \overline{x}$ dom$(x) = V_x$ with $|V_x| = N^{p_x/q}$ and build the instance as in the lowerbound proof of 3.1. 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)
- $|r_i^s| = N^{v_{y_i}} \leq N$
- $|r_i^e| = N^{v_{z_i}} \leq N$

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

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

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 $|R_{a_i}| = |r_j^s| = |r_j^e| = N$. $\square$

Unfortunately, not every combination of cardinalities of relations and vertices can be shown to produce tight bounds. However, as in (Atserias et al., 2013), 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 Eval$(Q, G')| \geq 2^{\rho^*(Q,G)-n}$, where n is the number of attributes of Q. As for full join queries, this is essentially the best we can get.

3.3. Bound for arbitrary ε-free CRPQs

Gottlob et al. study how to go from full join queries to conjunctive queries (Gottlob et al., 2012), 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).

The idea is to proceed in two steps. Consider a CRPQ with projection of the form

$$P(\overline{x_0}) = \pi_{\overline{x_0}} Q(\overline{x}). \tag{3.5}$$

We will first show how to transform the (full) Q into a simple CRPQ Q', for which the size bounds are the same up to a constant depending on Q. Afterwards, we construct, from the program of Q', a program for P that will provide our final bounds.

From full to simple CRPQs. The translation from a full CRPQ Q to a simple CRPQ Q' is quite direct. 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 $r = (a_1|a_2|\dots|a_k)$ (i.e. an expression accepting only words of size 1), with an atom $R_r(y_i, z_i)$, where R_r is a fresh relation. The following result is then shown using Gottlob et al's techniques (Gottlob et al., 2012).

Proposition 3.1 (full CRPQs)**.** *Consider a full CRPQ of form (2.1) in which every r_i is ε-free. For this query we have that $|Eval(Q, G)| \leq 2^{\rho^*(Q,G)}$. 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.*

PROOF. For the upper bound, given a CRPQ Q and a graph G, we transform Q into the simple query Q' as explained in Section 3.3, and generate from G 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. We see that $Q(G) = Q'(G')$ and Q' is a query that satisfies the conditions of Theorem 3.1 so $\text{Eval}(Q', G') \leq 2^{\rho^*(Q',G')}$. Also we have that $\rho^*(Q, G) = \rho^*(Q', G')$ so $\text{Eval}(Q, G) = 2^{\rho^*(Q,G)}$.

For the lower bound, as Q' suffices conditions of Theorem 3.1 (and Corollary 3.1), there are arbitrary graph database instances G' such that $\text{Eval}(Q', G') = 2^{\rho^*(Q',G')}$, 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. We will now build a graph database instance G compatible with Q that is exactly like G' but with the repeated relation defined as the union of the fresh relations of G'. Clearly we have that $\text{Eval}(Q, G) \geq \text{Eval}(Q', G') = 2^{\rho^*(Q',G')}$. On the other hand, as the relations and endpoint with weight greater than 0 all have the same size and the optimal cover for Q' is also a valid cover for Q we know 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. Finally, $|\text{Eval}(Q,G)| \cdot 2^{rep(Q)} \geq 2^{\rho^*(Q,G)}$. $\square$

Bounds for projections of simple CRPQs. We consider now queries of the form $P(\overline{x_0}) = \pi_{\overline{x_0}}Q(\overline{x})$, with $\overline{x_0} \subseteq \overline{x}$ noting that our previous result allows us to focus only on the case when Q is a simple CRPQ. As in (Gottlob et al., 2012), we consider a relaxation of the linear program for Q, in which we only keep those restrictions that refer to vertices of Q that are included in P. If we let $2^{\rho^*(P,G)}$ be the optimal solution of this program, we have

Proposition 3.2 (Queries with projections (Gottlob et al., 2012))**.** *Given an CRPQ P of the form (3.5) then for every graph database instance G we have that $|Eval(P,G)| \leq 2^{\rho^*(P,G)}$. Moreover, there are arbitrarily large instances G such that $|Eval(P,G)| = 2^{\rho^*(P,G)}$*

PROOF. The upper bound is a direct consequence of Theorem 3.1. Let Q' be a query just like Q, but removing all the attributes that are not projected. Then, as Q' satisfies the conditions of Theorem 3.1, we have that $|\text{Eval}(P,G)| \leq |\text{Eval}(Q',G')| \leq 2^{\rho(Q',G')}$. The linear program for both P and Q' have the same restrictions, only differing in the objective function so it must be that $\rho(Q',G') \leq \rho(P,G)$.

In the case of the lower bound we aim to build an instance G for P based on the instance G' from Theorem 3.1 for Q'. To do this, we will take the instance G' and 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 $\star$ the node from the proof of Theorem 3.1. In this case, $|\text{Eval}(P,G)| = |\text{Eval}(Q',G')| = 2^{\rho^*(Q',G')} = 2^{\rho^*(P,G)}$ $\square$

3.4. Dealing with ε

As we have mentioned, the evaluation of the expression ε over a graph G contains the *diagonal* $D = \{(v,v) \mid v \in G\}$. 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 by adding vertex variables to the linear program (and that may explode to be quadratic in size of the start and end points of expressions), and on the other there is ε, which behaves more like a relation, albeit with additional *key* constraints to guarantee that pairs in ε only come out of the diagonal.

Incorporating ε. The strategy once again involves using flat queries, reducing bounds on CRPQs to bounds on classes of queries we know how to handle. Here, we take a CRPQ Q that may use ε, and we produce instead a conjunctive query with functional dependencies.

More precisely, assume that Q is a simple CRPQ in which some of the RPQs r_i may instead be ε. We define $\mathit{flat}^{\varepsilon}(Q) = (\mathit{flat}(Q), \mathrm{FD})$, where $\mathit{flat}(Q)$ is again a conjunctive query, but one that now is coupled with a set FD of functional dependencies. The transformation goes just as for simple CRPQs, except that whenever Q contains an atom $\varepsilon(y_i, z_i)$, we replace it with atom $D(y_i, z_i)$ and add to FD dependencies $y_i \to z_i$ and $z_i \to y_i$.

We can make any graph G compatible with $\mathit{flat}^{\varepsilon}(Q) = (\mathit{flat}(Q), \mathrm{FD})$ by including the relation $D = \{(v, v) \mid v \in G\}$. Notice then that every graph does satisfy the functional dependencies in $\mathit{flat}(Q)$. Moreover, it is easy to see that Lemma 3.1 continues to hold: we always have that $|\mathrm{Eval}(Q, G)| \leq |\mathrm{Eval}(\mathit{flat}(Q), G)|$. What we gain with this transformation is that conjunctive queries with such simple functional dependencies are well understood (see (Gottlob et al., 2012; Abo Khamis, Ngo, & Suciu, 2016)), and for every query Q with *keys* (such as ours), we can find tight bounds. To state our result, we use $\mathrm{GLLV}(Q, \mathrm{FD}, G)$ to denote the output bounds devised for conjunctive queries with functional dependencies. We then have:

Proposition 3.3. *Let Q be a simple CRPQ in which some path queries may instead be the RPQ ε, and G a graph. Then $|Eval(Q, G)| \leq GLLV(\mathit{flat}^{\varepsilon}(Q), G)$. Moreover, there are arbitrarily big instances where this bound is tight.*

PROOF. For the upper bound, all we need to show is an extension of Lemma 3.1. Clearly, every graph G will always satisfy the functional dependencies in $\mathit{flat}^{\varepsilon}(Q)$. So the only thing left to show is that the fact that $(\mu(y_i), \mu(z_i))$ belong to $[\varepsilon]_G$ implies that $(\mu(y_i), \mu(z_i))$ belong to D, but this is by definition.

For the lower bound, we refer to (Gottlob et al., 2012), Proposition 4.5, where the authors show that for each pair (Q, FD), for Q a CQ without self-joins and FD a set of functional dependencies $x \to Y$ in which the left part is a single variable, and for each $N_0 \in \mathbb{N}$, there is a relational instance I in which all relations have at most $N \geq N_0$ tuples, and such that $\text{Eval}(Q, I) \geq \text{GLVV}(Q, \text{FD}, I)$. Now notice that, from such an instance, we can do the following.

- Replace all values appearing in the relation T in said construction with values $1, \dots, N$. Since we are equating some variables, this can only increase the number of answers we get and does not alter $\text{GLVV}(Q, \text{FD}, I)$.
- If, after GLVV's construction, relations D do not have all pairs $(i, i), i \leq N$, we add the remaining pairs. Again, this can only increase the number of answers we get, and does not alter $\text{GLVV}(Q, \text{FD}, I)$.

At the end we have an instance I realising the bound, where all elements are in $\{1, \dots, N\}$. Going from this instance (compatible with *flat*$^{\varepsilon}(Q)$) to a graph G compatible with Q that realizes the bounds is simple: all we need to do is to follow the construction in the lower bound of Theorem 3.1. More precisely, for each pair $r^s(y)$ and $r^e(z)$ in *flat*$^{\varepsilon}(Q)$, we take a path $w_1, \dots, w_n$, $n \geq 2$ in the language of r, and add to G edges $(i, w_1, 1)$ for each $i \in r^s$, $(1, w_j, 1)$ for $1 < j < n$ and $(1, w_n, p)$ for $p \in r^e$. we have that $r^s \times r^e \subseteq [r]$, from which we again have that $|\text{Eval}(Q, G))| \geq |\text{Eval}(\textit{flat}^{\varepsilon}(Q), G))| \geq \text{GLVV}(Q, \text{FD}, I)$, and where all cardinalities of relations and/or endpoints in G have arity at most N. This finishes the proof of the lower bound. $\square$

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 CRPQs. Consequently, we will look into *splitting* CRPQs into parts with ε and parts without it. For a CRPQ Q, let $r_{\ell_1}, \dots, r_{\ell_p}$ be the RPQs accepting ε. We define the family of queries $Q[S]$, for $S \subseteq \{\ell_1, \dots, \ell_p\}$, as follows. For each $r_\ell, \ell \in \{\ell_1, \dots, \ell_p\}$, 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 a fresh relation symbol K_ℓ if $\ell \notin S$.

Now augment G to make it compatible with any $Q[S]$ by letting $K_\ell = \{(a, a) \mid a \notin r^s_\ell\}$ for every $\ell \in \{\ell_1, \dots, \ell_p\}$. It is not too difficult to prove that $|\text{Eval}(Q, G)| \leq \sum_{S \subseteq \{\ell_1, \dots, \ell_p\}} |\text{Eval}(Q[S], G)|$, which yields our upper bound:

Proposition 3.4. *Let Q be a CRPQ. For any graph G,*

$$|\mathit{Eval}(Q, G)| \leq \sum_{S \subseteq \{\ell_1, \dots, \ell_p\}} \mathit{GLLV}(\mathit{flat}^\varepsilon(Q[S]), G).$$

Moreover, there are arbitrarily large graphs for which this bound is tight.

PROOF. For the lower bound, we proceed in a similar fashion to what we did for Proposition 3.3. Consider query $Q[S]$ where $S = \{\ell_1, \dots, \ell_p\}$, which is ε-free, and invoke the lower bound of corollary 3.1, 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 $\{1, \dots, N\}$
- For every remaining RPQ in Q where either r^s or r^e do not receive weights in the program, do $r^s = r^e = \{1, \dots, N\}$ (one can always do this by increasing the elements in one of the relations labelling the suitable edges in the automaton of r). Pick a word $w_1, \dots, w_n$ in the language of r and also add to G' edges $(i, w_1, 1)$ for each $i \in r^s$, $(1, w_j, 1)$ for $1 < j < n$ and $(1, w_n, p)$ for $p \in r^e$.

Since going from G to G' we are equating and adding elements, 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 $\text{Eval}(Q[S], G)$. To construct such set S, consider atom $r_{l_j}(y_{l_j}, z_{l_j})$, 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, t') belong to $[\hat{r}_{l_j}]_G$, 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 $\text{Eval}(Q[S], G)$. This is immediate for any atom not of the form $r_\ell(y_\ell, z_\ell)$ for $\ell \in \{\ell_1, \ldots, \ell_p\}$. For those atoms, note that $Q[S]$ has $\hat{r}_{l_j}(y_{l_j}, z_{l_j})$ whenever (t, t') belong to $[\hat{r}_{l_j}]_G$, so in this case μ also agrees with $\hat{r}_{l_j}(y_{l_j}, z_{l_j})$; and $Q[S]$ has $\varepsilon(y_{l_j}, z_{l_j})$ whenever (t, t') do not belong to $[\hat{r}_{l_j}]_G$. But if (t, t') do not belong to $[\hat{r}_{l_j}]_G$, since (t, t') is in $[r_{l_j}]_G$, it must be the case that $t = t'$ and $(t, t') \in [\varepsilon]_G$, which was to be shown. $\square$

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. ALGORITHMS TO EVALUATE CRPQS

In this chapter we try to see if any algorithm running in the worst-case optimal bound from Theorem 3.1 (and subsequent generalizations) exists. We call such an algorithm worst-case optimal, or WCO algorithm for short. The first result we obtain is that WCO algorithms might not even exist for CRPQs. 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 (Ngo et al., 2013)) on these materialized relations. We show that a modification of the GENERICJOIN algorithm of (Ngo et al., 2013) can approach the optimal performance of our baseline for many CRPQs.

4.1. WCO algorithms for CRPQs may not exist

Casel and Schmid show lower bounds for the problem of evaluating a single RPQ (Casel & Schmid, 2021). Specifically, for a graph $G = (V, E)$, and a (regular path) query $Q(x, y) = r(x, y)$, 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 (Williams & Williams, 2018), 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 problem? Unfortunately, this is not the case. To see this, consider again query $Q(x, y, z) = R_a(x, y) \wedge S_b(y, z) \wedge r(x, z)$, where r is any regular expression. Given a graph G in which $|R_a| = |S_b| = n$, our results tells 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 Eval(Q, G') if and only if (v_i, v_j) is the answer to r over G.

Proposition 4.1. *Any algorithm that, on input any arbitrary CRPQ Q and graph G, outputs Eval(Q, G) in time $O(2^{\rho^*(Q,G)})$, refutes the combinatorial 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:

(i) Compute the answer of each RPQ r appearing in Q over G.
(ii) Materialize all of these binary relations and add them to G.
(iii) Invoke a (relational) WCO algorithm (e.g. GENERICJOIN (Ngo et al., 2013)) 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 regular expressions.

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. (Arroyuelo, Hogan, Navarro, & Rojas-Ledesma, 2021), but we take a different approach.

In what follows, we impose that algorithms may only use memory in $O(V)$, for graphs (V, E). Since Proposition 4.1 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 AGM(Q, G) we denote the bound for maximal size of 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 $O(|V|^3 + \text{AGM}(Q, G))$, 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 Algorithm 1, so that it processes regular expressions 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 $Q(\overline{x}) = \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge \bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$ is a full CRPQ, and G a graph database, then Algorithm 2 defines GENERICJOINCRPQ(Q, G), a generalization of the GENERICJOIN WCO algorithm from the relational setting to graphs and (full) CRPQs. Similarly as in (Ngo et al., 2013), 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 (lines 8–10 and 12–14), add these results to our database (lines 11 and

Algorithm 2 GenericJoinCRPQ(Q, G)

```
1: $A \leftarrow \emptyset$
2: if $|\overline{x}| = 1$ then
3:     return Eval$(Q, G)$
4: Pick a variable $x \in \overline{x}$
5: $L \leftarrow \bigcap_{R(x,z) \in Q} R^s \bigcap_{R(y,x) \in Q} R^e \bigcap_{r(x,z) \in Q} r^s \bigcap_{r(y,x) \in Q} r^e$
6: for $v \in L$ do
7:     $\hat{Q} \leftarrow Q[x/v]$, $\hat{G} \leftarrow G$
8:     for each atom $r(x, z) \in Q$ do
9:         $r[v] \leftarrow \{v' \mid (v, v') \in [r]_G\}$.
10:        Replace $r(x, z)$ in $\hat{Q}$ for $r[v](z)$
11:        $\hat{G} \leftarrow G \cup r[v]$
12:    for each atom $r(y, x) \in Q$ do
13:        $r[v] \leftarrow \{v' \mid (v', v) \in [r]_G\}$
14:        Replace $r(y, x)$ in $\hat{Q}$ for $r[v](y)$
15:        $\hat{G} \leftarrow G \cup r[v]$
16:    $A[v] \leftarrow$ GenericJoinCRPQ$(\hat{Q}, \hat{G})$
17:    $A \leftarrow A \cup \{v\} \times A[v]$
18: return $Q$
```

15), and proceed recursively (line 16). The base case is when we have only one variable, in which case, we simply complete the missing values (line 3).

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, $Q_3(x, y, z) = a^+(x, y) \wedge b^+(y, z) \wedge R_c(x, z)$ as in Figure 1.3, and consider a graph G in which $|R_c| = M$ and all starting and ending points of RPQs (that is, R and S) have cardinality N. Here FULLMATERIALIZATION runs in time $O(N^3 + MN)$, 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 $L = S^s \cap R^e$. 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 $\hat{Q}(x, v, z) = a^+[v](x) \wedge b^+[v](z) \wedge R_c(x, z)$ over the augmented graph $\hat{G}$. This is an acyclic CQ, and GENERICJOINCRPQ($\hat{Q}, \hat{G}$) now defaults to GENERICJOIN($\hat{Q}, \hat{G}$), which solves this in time $O(M)$. Thus, the total running time is in $O(|L| \cdot (N^2 + M)) = O(N \cdot (N^2 + M))$. 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. Unfortunately, we cannot avoid it for all queries. Let us consider the triangle query but now with three RPQs: $Q(x, y, z) = a^+(x, y) \wedge b^+(y, z) \wedge c^+(x, z)$. 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 $\hat{Q}(x, v, z) = a^+[v](x) \wedge b^+[v](z) \wedge c^+(x, z)$ 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 $O(\text{AGM}(Q, G) \cdot |V|^2)$ 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 $Q(\overline{x}) = \bigwedge_{i=1}^{\ell} R_{a_i}(y_i, z_i) \wedge$

Algorithm 3 GenericJoinCRPQ-Bipartite$(Q, G, \overline{x}_1)$

```
 1: $A \leftarrow \emptyset$
 2: **if** $|\overline{x}| = 1$ **then**
 3:     **return** Eval$(Q, G)$
 4: $L \leftarrow \mathsf{GenericJoin}(Q_{\overline{x}_1}, G)$
 5: **for** $\mathbf{t}_{\overline{x}_1} \in L$ **do**
 6:     **for** $i \in [\ell + q, k]$ **do**
 7:         **if** $y_i \in \overline{x}_1$ **then**                              ▷ processing $r_i(y_i, z_i)$
 8:             $r_i[v] \leftarrow \{v' \mid (v, v') \in [r_i]_G\}$
 9:             Replace $r_i(y_i, z_i)$ in $\hat{Q}$ for $r_i[v](z_i)$
10:         **else**                                         ▷ bipartite implies $z_i \in \overline{x}_1$
11:             $r_i[v] \leftarrow \{v' \mid (v', v) \in [r_i]_G\}$
12:             Replace $r_i(y_i, z_i)$ in $\hat{Q}$ for $r_i[v](y_i)$
13:         $\hat{G} \leftarrow G \cup r_i[v]$
14:     $Q[\mathbf{t}_{\overline{x}_1}] \leftarrow \mathsf{GenericJoin}(\hat{Q}, \hat{G})$
15:     $Q \leftarrow Q \cup \{\mathbf{t}_{\overline{x}_1}\} \times Q[\mathbf{t}_{\overline{x}_1}]$
16: **return** $Q$
```

$\bigwedge_{i=\ell+1}^{k} r_i(y_i, z_i)$. 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 $E_r = \{(y_i, z_i) \mid i = \ell + 1, \ldots, k\}$, 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 $\overline{x}_1 \subseteq \overline{x}$ be a bipartiton of the RPQ-graph of Q. Then evaluating Q over a graph database G can be done via Algorithm 3, 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 $\overline{x}_1$.

Algorithm GENERICJOINCRPQ-BIPARTITE generalizes Algorithm 2 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:

(i) First, we compute the tuples $\mathbf{t}_{\overline{x}_1}$ in the answer of $Q_{\overline{x}_1}$ using the relational Algorithm 1 (line 4)[1].

(ii) For every tuple $\mathbf{t}_{\overline{x}_1}$ we compute all the associated regular expressions (lines 5–13).

(iii) We compute the rest of the join with the relational GenericJoin (line 14).

In the worst case, we need to perform AGM$(Q_{\overline{x}_1}, G_{\overline{x}_1})$ computations of every regular expression r_i. Therefore, the total cost will be in $O(\text{AGM}(Q_{\overline{x}_1}, G_{\overline{x}_1}) \times |V|^2)$ (the $|V|^2$ being the cost of computing the RPQs). On the other hand the result size of Q over G is bounded by $O(\text{AGM}(Q, G))$, so the final join might reach this bound. Of course, in order to minimize the computation time, we will always select $\overline{x}_1$ to be the smaller bipartition. Thus, we obtain the following.

Theorem 4.1. *Let Q be a CRPQ such that its RPQ-graph is bipartite, and let $V_Q = V'_Q \cup V''_Q$ be an RPQ-bipartition. Then the running time of* GENERICJOINCRPQ-BIPARTITE *over Q is*

$$O(AGM(Q, G) + \min\{AGM(Q', G'), AGM(Q'', G'')\} \cdot |V|^2)$$

with Q' and Q'' the CRPQs restricted to conjuncts using only the variables in V' and V'' respectively.

PROOF. Following the decomposition of the algorithm we did above, 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 AGM(Q_1, G_1) with Q_1 query Q limited to the attributes in $\overline{x}_1$. Then for every tuple in L, we compute each of the RPQs of Q. The cost of every computation is in $O(|V|^2)$. In total, the cost of computing the RPQs is in $O(AGM(Q_1) \cdot |V|^2)$. After computing and updating the graph, the cost of the

[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.

computing the join of the remaining part of the query is in $O(\text{AGM}Q_2)$ with Q_2 the query restricted to the attributes in $\overline{x} - \overline{x}_1$. However, we can bound both the join in line 4 and the ones in line 16 by the overall cost of computing the whole join via generic algorithm. Adding up we get a running time in

$$O(\text{AGM}(Q) + \min\{\text{AGM}(Q'), \text{AGM}(Q'')\} \cdot |V|^2)$$

where the part of the left is the cost of computing the query as if it had no RPQs and the left part is the cost of actually computing the RPQs. $\square$

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 4.1. *When the RPQ-graph of a CRPQ Q is bipartite and it admits a partition $V_Q = V'_Q \cup V''_Q$ with $\min\{|V'|, |V''|\} = 1$, 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

This thesis provides techniques for understanding size bounds of CRPQs, and makes use of these techniques to inform better algorithms for evaluating CRPQs. We proved that while the AGM bounds developed for relational queries can be adapted to CRPQs on graph databases, the worst-case optimal algorithms for those queries lose their worst-case optimality condition when extended to 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.

On the other hand, 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

Abo Khamis, M., Ngo, H. Q., & Suciu, D. (2016). Computing join queries with functional dependencies. In *Proceedings of the 35th acm sigmod-sigact-sigai symposium on principles of database systems* (pp. 327–342).

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J. L., & Vrgoc, D. (2017). Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.*, *50*(5), 68:1–68:40.

Arroyuelo, D., Hogan, A., Navarro, G., Reutter, J. L., Rojas-Ledesma, J., & Soto, A. (2021). Worst-case optimal graph joins in almost no space. In G. Li, Z. Li, S. Idreos, & D. Srivastava (Eds.), *SIGMOD '21: International conference on management of data, virtual event, china, june 20-25, 2021* (pp. 102–114). ACM.

Arroyuelo, D., Hogan, A., Navarro, G., & Rojas-Ledesma, J. (2021). Time-and space-efficient regular path queries on graphs. *arXiv preprint arXiv:2111.04556*.

Atserias, A., Grohe, M., & Marx, D. (2013). Size bounds and query plans for relational joins. *SIAM J. Comput.*, *42*(4), 1737–1767.

Baeza, P. B. (2013). 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* (pp. 175–188).

Bonifati, A., Martens, W., & Timm, T. (2020). An analytical study of large SPARQL query logs. *VLDB J.*, *29*(2-3), 655–679.

Casel, K., & Schmid, M. L. (2021). Fine-grained complexity of regular path queries. In *24th international conference on database theory, ICDT 2021, march 23-26, 2021, nicosia, cyprus* (pp. 19:1–19:20).

Freitag, M. J., Bandle, M., Schmidt, T., Kemper, A., & Neumann, T. (2020). Adopting worst-case optimal joins in relational database systems. *Proc. VLDB Endow.*, *13*(11), 1891–1904.

Gottlob, G., Lee, S. T., Valiant, G., & Valiant, P. (2012). Size and treewidth bounds for conjunctive queries. *J. ACM*, *59*(3), 16:1–16:35.

Hogan, A., Riveros, C., Rojas, C., & Soto, A. (2019). 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* (pp. 258–275).

Neumann, T., & Weikum, G. (2008). Rdf-3x: a risc-style engine for rdf. *Proceedings of the VLDB Endowment*, *1*(1), 647–659.

Neumann, T., & Weikum, G. (2010). The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, *19*(1), 91–113.

Ngo, H. Q., Porat, E., Ré, C., & Rudra, A. (2012). Worst-case optimal join algorithms. In *PODS 2012* (pp. 37–48).

Ngo, H. Q., Ré, C., & Rudra, A. (2013). Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, *42*(4), 5–16.

Nguyen, D., Aref, M., Bravenboer, M., Kollias, G., Ngo, H. Q., Ré, C., & Rudra, A. (2015). Join processing for graph patterns: An old dog with new tricks. In *Proceedings of the grades'15* (pp. 1–8).

Team, J. (2021). *TDB Documentation.* Retrieved from `https://jena.apache.org/documentation/tdb/`

Veldhuizen, T. L. (2014). Triejoin: A simple, worst-case optimal join algorithm. In N. Schweikardt, V. Christophides, & V. Leroy (Eds.), *Proc. 17th international conference* *on database theory (icdt), athens, greece, march 24-28, 2014* (pp. 96–106). OpenProceedings.org.

Williams, V. V., & Williams, R. R. (2018). Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, *65*(5), 1–38.

Wood, P. T. (2012). Query languages for graph databases. *SIGMOD Rec.*, *41*(1), 50–60.