Fine-Grained Complexity of Regular Path Queries

Katrin Casel
Hasso Plattner Institute, Universität Potsdam, Germany

Markus L. Schmid
Humboldt-Universität zu Berlin, Germany

Abstract

A regular path query (RPQ) is a regular expression q that returns all node pairs (u, v) from a graph database that are connected by an arbitrary path labelled with a word from $L(q)$. The obvious algorithmic approach to RPQ evaluation (called PG-approach), i. e., constructing the product graph between an NFA for q and the graph database, is appealing due to its simplicity and also leads to efficient algorithms. However, it is unclear whether the PG-approach is optimal. We address this question by thoroughly investigating which upper complexity bounds can be achieved by the PG-approach, and we complement these with conditional lower bounds (in the sense of the fine-grained complexity framework). A special focus is put on enumeration and delay bounds, as well as the data complexity perspective. A main insight is that we can achieve optimal (or near optimal) algorithms with the PG-approach, but the delay for enumeration is rather high (linear in the database). We explore three successful approaches towards enumeration with sub-linear delay: super-linear preprocessing, approximations of the solution sets, and restricted classes of RPQs.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness; Theory of computation → Database query languages (principles); Theory of computation → Data structures and algorithms for data management

Keywords and phrases Graph Databases, Regular Path Queries, Enumeration, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2021.19

Related Version *Full Version*: `https://arxiv.org/abs/2101.01945`

Funding *Katrin Casel*: Supported by the Federal Ministry of Education and Research of Germany (BMBF) in the KI-LAB-ITSE framework – project number 01IS19066.
Markus L. Schmid: Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735).

Acknowledgements We wish to thank the anonymous reviewers for their valuable feedback. In particular, following the reviewer's comments and suggestions, we have included more background information and comprehensive explanations of certain aspects, which substantially improved the overall exposition of this paper.

1 Introduction

An essential component of graph query languages (to be found both in academical prototypes as well as in industrial solutions) are *regular path queries* (RPQs). Abstractly speaking, a regular expression q over some alphabet Σ is interpreted as query that returns from a Σ-edge-labelled, directed graph $\mathcal{D}$ (i. e., a *graph database*) the set $q(\mathcal{D})$ of all node pairs (u, v) that are connected by a q-path, i. e., a path labelled with a word from q's language (and possibly also a witness path per node pair, or even all such paths). This simple, yet relevant concept has heavily been studied in database theory (the following list is somewhat biased towards recent work): results on RPQs [26, 7, 35, 37, 40, 16], conjunctive RPQs [15, 43, 9] and extensions thereof [34, 12, 34, 29], questions of static analysis [28, 11, 27, 31, 44], experimental analyses [17, 19, 39], and surveys of this research area [10, 53, 5, 23].

24th International Conference on Database Theory (ICDT 2021).
Editors: Ke Yi and Zhewei Wei; Article No. 19; pp. 19:1–19:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the simplest setting, where we are only interested in the node pairs (but no paths) connected by *arbitrary* q-paths (instead of, e. g., simple paths), evaluation can be done efficiently. Deviating from this simple setting, however, leads to intractability: if we ask for nodes connected by *simple paths* (no repeated nodes), or connected by *trails* (no repeated arcs), then RPQ evaluation is NP-hard even in data-complexity (see [41, 7, 38] and [37], respectively). Note that the simple path and trail semantics are mostly motivated by the fact that under these semantics there is only a finite number of q-paths per node pair. If we move to conjunctions of RPQs (CRPQs) or even more powerful extensions motivated by practical requirements, then also with the arbitrary path semantics evaluation becomes intractable in combined complexity (i. e., they inherit hardness from relational conjunctive queries (CQs)).

In order to guide practical developments in the area of graph databases, the computational hard cases of RPQ (and CRPQ) evaluation have been thoroughly investigated in database theory. However, with respect to arbitrary q-paths, research seems to have stopped at the conclusion that efficient evaluation is possible by the following simple *PG-approach*: given graph database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ and NFA M_q for q with state set V_q, construct the *product graph* $G_{\mathcal{D},q}$ with nodes $V_{\mathcal{D}} \times V_q$ and an arc $((u, p), (v, p'))$ iff, for some $\mathsf{a} \in \Sigma$, $\mathcal{D}$ has an arc (u, a, v) and M_q has an arc (p, a, p'), and then use simple graph-searching techniques on $G_{\mathcal{D},q}$.

The PG-approach is explicitly defined in several papers, e. g., [41, 10, 38], and mainly used to prove a worst-case upper bound (actually $\mathrm{O}(|q||\mathcal{D}|)$ for Boolean evaluation); in [38] it is used for enumerating q-paths between two given nodes. But it is also very appealing from a practical point of view due to its simplicity: we are just coupling well-understood algorithmic concepts like finite automata and graph reachability algorithms. Arguably, implementing the PG-approach is an exercise suitable for a first year programming course (making it feasible and cost-efficient for industrial systems). As it seems, putting RPQ evaluation with arbitrary path semantics and the respective PG-approach into the focus of a thorough theoretical study has not yet been done. This paper is devoted to this task. In particular, we wish to investigate the following two (somewhat overlapping) aspects:

1. Applicability of the PG-approach: the PG-approach is suited for solving simple evaluation problems like checking $q(\mathcal{D}) = \emptyset$ or $(u, v) \in q(\mathcal{D})$ (for given $u, v \in V_{\mathcal{D}}$), but is it also appropriate for more relevant tasks like computing, counting or enumerating $q(\mathcal{D})$?
2. Optimality of the PG-approach: Does the PG-approach lead to optimal algorithms, or can it be beaten by conceptionally different techniques?

Answering these questions provides a better theoretical understanding of RPQ-evaluation (which, as mentioned above, are at the heart of many graph query languages). But also for the more powerful CRPQs and more practically motivated graph query languages, we can derive valuable insights from our investigation. Let us mention two such examples (a complete summary of our results follows further down). As noted in [10], we can reduce CRPQ evaluation to the evaluation of relational CQs by first constructing all tables represented by the single RPQs and then evaluating a CQ over this database. To do this, we first have to compute the results of all RPQs, so it seems helpful to know the best algorithms for this intermediate task. Moreover, if we want to benefit from the existing CQ evaluation techniques (e. g., exploiting acyclicity etc.) we are more or less forced to this two-step approach. With respect to *enumerations* of CQs, it is known that linear preprocessing and constant delay enumeration is possible provided that the CQs satisfy certain acyclicity properties (see [8, 14], or the surveys [13, 45]). Unfortunately, these techniques do not carry over to CRPQs since, as we show, linear preprocessing and constant delay enumeration is not possible even for single RPQs (conditional to some complexity assumptions).

Table 1 All upper bounds can be achieved as running times of some algorithm, while the lower bounds cannot be achieved as running time by any algorithm, unless the displayed hypothesis fails. The exponent ω denotes the best known matrix multiplication exponent.

Non-enum. Results		Boole, Test, Witness	Eval	Count
upper bounds		$\mathrm{O}(\lvert\mathcal{D}\rvert\lvert q\rvert)$	$\mathrm{O}(\lvert V_{\mathcal{D}}\rvert\lvert\mathcal{D}\rvert\lvert q\rvert)$ $\mathrm{O}((\lvert V_{\mathcal{D}}\rvert\lvert q\rvert)^{2.37})$	$\mathrm{O}(\lvert V_{\mathcal{D}}\rvert\lvert\mathcal{D}\rvert\lvert q\rvert)$ $\mathrm{O}((\lvert V_{\mathcal{D}}\rvert\lvert q\rvert)^{\omega})$
lower bounds	OV & com-BMM	$\mathrm{O}((\lvert\mathcal{D}\rvert\lvert q\rvert)^{1-\epsilon})$	—	—
	OV	—	—	$\mathrm{O}^{\mathsf{dc}}((\lvert V_{\mathcal{D}}\rvert\lvert\mathcal{D}\rvert)^{1-\epsilon})$
	SBMM	—	$\mathrm{O}^{\mathsf{dc}}(\lvert q(\mathcal{D})\rvert+\lvert\mathcal{D}\rvert)$	—
	com-BMM	—	$\mathrm{O}^{\mathsf{dc}}((\lvert V_{\mathcal{D}}\rvert\lvert\mathcal{D}\rvert)^{1-\epsilon})$	—

Since the problem we investigate can be solved in polynomial time (also in combined complexity), we cannot show lower bounds in terms of hardness results for complexity classes like NP or PSPACE. Instead, we make use of the framework of *fine-grained complexity*, which allows to prove lower bounds that are conditional on some algorithmic assumptions (see the surveys [50, 22, 51]). In particular, fine-grained complexity is a rather successful toolbox for giving evidence that the *obvious* algorithmic approach to some basic problem, is also the *optimal* one. This is exactly our setting here, with respect to RPQ-evaluation and the PG-approach. To the knowledge of the authors, such conditional lower bounds are not yet a well-established technique in database theory (however, see Section [13, Section 6] for a survey of conditional lower bounds in the context of CQ enumeration).

A main challenge is that fine-grained complexity is not exactly tailored to either the data-complexity perspective or to enumeration problems. We will next outline our results.

1.1 Our Contribution

All investigated RPQ-evaluation problems are summarised on page 7. In the following, $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ is the graph database, q is the RPQ, and $\epsilon > 0$. With the notation $\mathrm{O}^{\mathsf{dc}}(\cdot)$, we hide factors $f(|q|)$ for some function f (i. e., it is used for stating data-complexities). All lower bounds mentioned in the following are conditional to some of the algorithmic assumptions summarised in Section 4 (we encourage the reader less familiar with fine-grained complexity hypotheses to have a look at this section first, which can be read independently). For presentational reasons, we do not always explicitly mention this in the rest of the introduction and when we say that a certain running time is "not possible", this statements is always conditional in this sense (see Tables 1 and 2 for the actual hypotheses). As common in fine-grained complexity, we rule out *true* sub-linear ($\mathrm{O}(n^{1-\epsilon})$), sub-quadratic ($\mathrm{O}(n^{2-\epsilon})$), or sub-cubic ($\mathrm{O}(n^{3-\epsilon})$) running times, but not possible improvements by logarithmic factors.

Non-Enumeration Variants

The following results are summarised in Table 1. For the simple problems Boole (checking $q(\mathcal{D}) = \emptyset$), Test (checking $(u, v) \in q(\mathcal{D})$) and Witness (computing some element from $q(\mathcal{D})$), the PG-approach yields an upper bound of $\mathrm{O}(|\mathcal{D}||q|)$, which is optimal (since linear) in data complexity, and we can show lower bounds demonstrating its optimality also in combined complexity. For Eval (computing the set $q(\mathcal{D})$) the PG-approach yields a data complexity upper bound of $\mathrm{O}^{\mathsf{dc}}(|V_{\mathcal{D}}||\mathcal{D}|)$, which cannot be improved by *combinatorial* algorithms, although $\mathrm{O}^{\mathsf{dc}}(|V_{\mathcal{D}}|^{2.37})$ is possible by fast matrix multiplication (see Section 4 for a discussion of the meaning of the term "combinatorial"). In addition, we can show that

Table 2 All upper bounds can be achieved as running times of some algorithm, while the lower bounds cannot be achieved as running time by any algorithm, unless the displayed hypothesis fails. The exponent ω denotes the best known matrix multiplication exponent.

Enum. Results		Enum			
		preprocessing	delay	sorted	updates
upper bound		$\mathrm{O}(\text{delay})$	$\mathrm{O}(\lvert\mathcal{D}\rvert\lvert q\rvert)$	✓	$\mathrm{O}(1)$
lower bounds	OV & com-BMM	$\mathrm{O}(\text{delay})$	$\mathrm{O}((\lvert\mathcal{D}\rvert\lvert q\rvert)^{1-\epsilon})$	×	×
	SBMM	$\mathrm{O}^{\mathsf{dc}}(\lvert\mathcal{D}\rvert)$	$\mathrm{O}^{\mathsf{dc}}(1)$	×	×
	com-BMM	$\mathrm{O}^{\mathsf{dc}}(\lvert\mathcal{D}\rvert)$	$\mathrm{O}^{\mathsf{dc}}(\lvert V_{\mathcal{D}}\rvert^{1-\epsilon})$	×	×
	OMv	arbitrary	$\mathrm{O}^{\mathsf{dc}}(\lvert V_{\mathcal{D}}\rvert^{1-\epsilon})$	×	$\mathrm{O}^{\mathsf{dc}}(\lvert V_{\mathcal{D}}\rvert^{1-\epsilon})$
	com-BMM	$\mathrm{O}^{\mathsf{dc}}(\lvert\mathcal{D}\rvert)$	$\mathrm{O}^{\mathsf{dc}}(\lvert V_{\mathcal{D}}\rvert^{2-\epsilon})$	×	$\mathrm{O}^{\mathsf{dc}}(\lvert V_{\mathcal{D}}\rvert^{2-\epsilon})$

linear time data complexity, i. e., $\mathrm{O}^{\mathsf{dc}}(|q(\mathcal{D})| + |\mathcal{D}|)$, is not possible even for non-combinatorial algorithms. For Count (computing $|q(\mathcal{D})|$), we get $\mathrm{O}^{\mathsf{dc}}(|V_{\mathcal{D}}||\mathcal{D}|)$ as upper and lower bound, not restricted to combinatorial algorithms.

Enumeration

Our results for RPQ-enumeration are summarised in Table 2. An entry "O(delay)" in column "preprocessing" means that the preprocessing is bounded by the delay (which means that no preprocessing is required). The column "sorted" indicates whether the enumeration is produced lexicographically sorted.

In comparison to the non-enumeration problem variants, the picture is less clear and deserves more explanation. The PG-approach yields a simple enumeration algorithm with delay $\mathrm{O}(|\mathcal{D}||q|)$, that also trivially supports updates in constant time, since the preprocessing fits into the delay bound. Our lower bounds for Boole also mean that this delay cannot be improved in terms of *combined complexity*. While this lower bound was interesting for problems like Boole etc., it now gives a correct answer to the wrong question. The main goal now should be to find out whether we can remedy the linear dependency of the delay on $|\mathcal{D}|$, at the expense of spending more time in terms of $|q|$, or of losing the ability of handling updates, or even of allowing a slightly super-linear preprocessing.

In this regard, the strongest result would be linear preprocessing $\mathrm{O}(|\mathcal{D}|f(|q|))$ and constant delay $\mathrm{O}(f(|q|))$. However, we can rule this out even for algorithms not capable of handling updates. Then, the next question is which non-constant delays can be achieved that are strictly better than linear. For example, none of our lower bounds for the non-enumeration variants suggest that linear preprocessing and a delay bounded by, e. g., $|V_{\mathcal{D}}|$ or the degree of $\mathcal{D}$, should not be possible. We are *not* able to answer this question in its general form (and believe it to be very challenging), but we are able to provide several noteworthy insights.

For linear preprocessing, a delay of $\mathrm{O}(|V_{\mathcal{D}}|)$ (if possible at all) cannot be beaten by combinatorial algorithms (even without updates). This can be strengthened considerably, if we also require updates in some reasonable time: for general algorithms (i. e., not necessarily combinatorial) delay *and* update time strictly better than $\mathrm{O}^{\mathsf{dc}}(|V_{\mathcal{D}}|)$ is not possible even with arbitrary preprocessing, and for combinatorial algorithms with linear preprocessing even delay *and* update time of $\mathrm{O}(|\mathcal{D}|)$ cannot be beaten. This last result nicely complements the upper bound at least for combinatorial algorithms and in the dynamic case.

In summary, for linear preprocessing, $\mathrm{O}(|V_{\mathcal{D}}|)$ is a lower bound for the delay and if we can beat $\mathrm{O}(|\mathcal{D}|)$, we should not be able to also support updates.

Enumeration of Restricted Variants

Finally, we obtain restricted problem variants that can be solved with delay strictly better than $\mathrm{O}(|\mathcal{D}|)$ (in data complexity). We explore three different approaches: (1) by allowing super-linear preprocessing of $\mathrm{O}^{\mathsf{dc}}(\overline{\Delta}(\mathcal{D})\log(\overline{\Delta}(\mathcal{D}))|\mathcal{D}|)$ (where $\overline{\Delta}(\mathcal{D})$ is the average degree of $\mathcal{D}$), we can achieve a delay of $\mathrm{O}(|V_{\mathcal{D}}|)$; (2) in linear preprocessing and constant delay, we can enumerate a representative subset of $q(\mathcal{D})$ instead of the whole set $q(\mathcal{D})$; (3) for a subclass of RPQs, we can solve RPQ-Enum with linear preprocessing and delay $\mathrm{O}(\Delta(\mathcal{D}))$ (where $\Delta(\mathcal{D})$ is the maximum degree of $\mathcal{D}$).

Due to space constraints, most proof details are omitted, although we give proof sketches for some results; detailed proofs can be found in the preliminary full version of this paper [25].

2 Main Definitions

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $[n] = \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. For a finite alphabet A, A^+ denotes the set of non-empty words over A and $A^* = A^+ \cup \{\varepsilon\}$ (where ε is the empty word). For a word $w \in A^*$, $|w|$ denotes its length;$w^1 = w$ and $w^k = ww^{k-1}$ for every $k \geq 2$. For $L, K \subseteq A^*$, let $L \cdot K = \{w_1 \cdot w_2 \mid w_1 \in L, w_2 \in K\}$, let $L^1 = L$ and $L^k = L \cdot L^{k-1}$ for every $k \geq 2$, let $L^+ = \bigcup_{k \geq 1} L^k$ and $L^* = L^+ \cup \{\varepsilon\}$.

2.1 Σ-Graphs

We now define the central graph model that is used to represent graph databases as well as finite automata. Let Σ be a finite alphabet of constant size. A *Σ-graph* is a directed, edge labelled multigraph $G = (V, E)$, where V is the set of *vertices* (or *nodes*) and $E \subseteq V \times (\Sigma \cup \{\varepsilon\}) \times V$ is the set of *edges* (or *arcs*). For $u \in V$ and $x \in \Sigma \cup \{\varepsilon\}$, $E_x(u) = \{v \mid (u, x, v) \in E\}$ is the set of *x-successors of u*. A path from $w_0 \in V$ to $w_k \in V$ of length $k \geq 0$ is a sequence $p = (w_0, a_1, w_1, a_2, w_2 \ldots, w_{k-1}, a_k, w_k)$ with $(w_{i-1}, a_i, w_i) \in E$ for every $i \in [k]$. We say that p is *labelled* with the *word* $a_1 a_2 \ldots a_k \in \Sigma^*$. According to this definition, for every $v \in V$, (v) is a path from v to v of length 0 that is labelled by ε. Hence, every node v of a Σ-graph has an ε-labelled path to itself, even though there might not be an ε-arc from v to v. Moreover, due to ε as a possible edge-label, paths of length k may be labelled with words w with $|w| < k$. The size of $G = (V, E)$ is $|G| = \max\{|V|, |E|\}$.

For any Σ-graph $G = (V, E)$, we call $(V, \{(u, v) \mid u \neq v \wedge \exists x \in \Sigma \cup \{\varepsilon\} : (u, x, v) \in E\})$ the *underlying graph* of G (note that the underlying graph is simple, non-labelled and has no loops). In particular, by a slight abuse of notation, we denote by E^* the reflexive-transitive closure of the underlying graph of G. Since we always assume $|\Sigma|$ to be a constant, we have that $|G| = \Theta(\max\{|V|, |\{(u, v) \mid u \neq v \wedge \exists x \in \Sigma \cup \{\varepsilon\} : (u, x, v) \in E\}|\})$ (i. e., $|G|$ is asymptotically equal to the size of its underlying graph). For every $u \in V$, the *degree of u* is $\Delta(u) = |\bigcup_{x \in \Sigma \cup \{\varepsilon\}} E_x(u)|$ (so $\Delta(u)$ is actually the out-degree), and the *maximum degree* of G is $\Delta(G) = \max\{|\Delta(u)| \mid u \in V\}$. The *average degree* of G is $\overline{\Delta}(G) = \frac{1}{|V|} \sum_{u \in V} |\Delta(u)|$. Obviously, $\overline{\Delta}(G) \leq \Delta(G) \leq |V|$.

Since Σ-graphs are the central data structures for our algorithms, we have to discuss implementational aspects of Σ-graphs in more detail. The set V of a Σ-graph $G = (V, E)$ is represented as a list, and, for every $u \in V$ and for every $x \in \Sigma \cup \{\varepsilon\}$, we store a list of all x-successors of u, which is called the *x-adjacency list* for u. We assume that we can check in constant time whether a list is empty and we can insert elements in constant time. However, finding and deleting an element from a list requires linear time. Furthermore, we assume that we always store together with a node a pointer to its adjacency list (thus, we can always retrieve the x-adjacency list for a given node in constant time).

▶ Remark 1. All lower bounds presented in this paper hold for any graph representation that can be constructed in time linear in $|G| = \max\{|V|, |E|\}$. For the upper bounds, we chose the simple representation with adjacency lists as it emerged as the natural structure for our enumeration approach; let us point out here that since we always store pointers to the adjacency lists along with the nodes, we can perform a breadth-first search (BFS) from any given start node u in time $\mathrm{O}(|G|)$. It is a plausible assumption that most specific graph representations can be transformed into our list-based representation without much effort. This ensures a certain generality of our upper complexity bounds in the sense that the corresponding algorithms are, to a large extent, independent from implementational details. Note also that the list-based structure only requires space linear in $|G|$.

In the adjacency list representation, we do not have random access to specific nodes in the graph database, or to specific neighbours of a given node. Thus, we have to measure a non-constant running-time for performing such operations. However, the algorithms for our upper bounds are independent from this aspect, i. e., the total running times would not change if we assume random access to nodes in constant time. Therefore, it is a stronger statement that a conditional lower bound is matched by an algorithm that *does not* exploit specific implementational aspects or advanced data structures, but works for basic graph representations.

An exception to this is Theorem 20, for which we can obtain some small improvement by applying the technique of lazy array initialization (see Remark 21).

2.2 Graph Databases and Regular Path Queries

A *nondeterministic finite automaton* (NFA for short) is a tuple $M = (G, S, T)$, where $G = (V, E)$ is a Σ-graph (the nodes $q \in V$ are also called *states*), $S \subseteq V$ with $S \neq \emptyset$ is the set of *start states* and $T \subseteq V$ with $T \neq \emptyset$ is the set of *final states*. The language $\mathcal{L}(M)$ of an NFA M is the set of all labels of paths from some start state to some final state. For a Σ-graph $G = (V, E)$, any subsets $S, T \subseteq V$ with $S \neq \emptyset \neq T$ induce the NFA (G, S, T). If $S = \{s\}$ and $T = \{t\}$ are singletons, then we also write (G, s, t) instead of $(G, \{s\}, \{t\})$.

The set RE_Σ of *regular expressions* (*over* Σ) is recursively defined as follows: $a \in \mathsf{RE}_\Sigma$ for every $a \in \Sigma \cup \{\varepsilon\}$; $(\alpha \cdot \beta) \in \mathsf{RE}_\Sigma$, $(\alpha \vee \beta) \in \mathsf{RE}_\Sigma$, and $(\alpha)^+ \in \mathsf{RE}_\Sigma$, for every $\alpha, \beta \in \mathsf{RE}_\Sigma$. For any $\alpha \in \mathsf{RE}_\Sigma$, let $\mathcal{L}(\alpha)$ be the regular language described by the regular expression α defined as usual: for every $a \in \Sigma \cup \{\varepsilon\}$, $\mathcal{L}(a) = \{a\}$, and for every $\alpha, \beta \in \mathsf{RE}_\Sigma$, $\mathcal{L}(\alpha \cdot \beta) = \mathcal{L}(\alpha) \cdot \mathcal{L}(\beta)$, $\mathcal{L}(\alpha \vee \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$ and $\mathcal{L}(\alpha^+) = \mathcal{L}(\alpha)^+$. We also use α^* as short hand form for $\alpha^+ \vee \varepsilon$. By $|\alpha|$, we denote the length of α represented as a string.

▶ **Proposition 2.** *Every regular expression α can be transformed in time $\mathrm{O}(|\alpha|)$ into an equivalent* NFA *$M = (G, p_0, p_f)$ with $|G| = \mathrm{O}(|\alpha|)$.*

In the following, when we speak about an *automaton* (or an NFA) *for a regular expression* α, we always mean an NFA equivalent to α with the properties asserted by Proposition 2.

A Σ-graph without ε-arcs is also called a *graph database* (*over* Σ); in the following, we denote graph databases by $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$. Since $V_\mathcal{D}$ is represented as a list, any graph database implicitly represents a linear order on $V_\mathcal{D}$ (i. e., the order induced by the list that represents $V_\mathcal{D}$), which we denote by $\preceq_\mathcal{D}$, or simply $\preceq$ if $\mathcal{D}$ is clear from the context. A graph database $\mathcal{D}$ is *sparse* if $|E_\mathcal{D}| = \mathrm{O}(|V_\mathcal{D}|)$.

Regular expressions q (over alphabet Σ) are interpreted as *regular path queries* (RPQ) for graph databases (over Σ). The *result* $q(\mathcal{D})$ of an RPQ q on a graph database $\mathcal{D} = (V_\mathcal{D}, E_\mathcal{D})$ over Σ is the set $q(\mathcal{D}) = \{(u, v) \mid u, v \in V_\mathcal{D}, \mathcal{L}((\mathcal{D}, u, v)) \cap \mathcal{L}(q) \neq \emptyset\}$.

If we interpret q as a *Boolean* RPQ, then the result is $q_{\mathsf{B}}(\mathcal{D}) = \mathsf{true}$ if $q(\mathcal{D}) \neq \emptyset$ and $q_{\mathsf{B}}(\mathcal{D}) = \mathsf{false}$ otherwise. We consider the following RPQ evaluation problems ($\mathcal{D}$ is a graph database, q an RPQ and u, v two nodes from $\mathcal{D}$):

Name	Input	Task
RPQ-Boole	$\mathcal{D}$, q	Decide whether $q_{\mathsf{B}}(\mathcal{D}) = \mathsf{true}$.
RPQ-Test	$\mathcal{D}$, q, u, v	Decide whether $(u, v) \in q(\mathcal{D})$.
RPQ-Witness	$\mathcal{D}$, q	Compute a witness $(u, v) \in q(\mathcal{D})$ or report that none exists.
RPQ-Eval	$\mathcal{D}$, q	Compute the whole set $q(\mathcal{D})$
RPQ-Count	$\mathcal{D}$, q	Compute $\lvert q(\mathcal{D})\rvert$.
RPQ-Enum	$\mathcal{D}$, q	Enumerate the whole set $q(\mathcal{D})$.

By *sorted RPQ-Enum* (or *semi-sorted* RPQ-Enum), we denote the variant of RPQ-Enum, where the pairs of $q(\mathcal{D})$ are to be enumerated in lexicographical order with respect to $\preceq_{\mathcal{D}}$ (or ordered only with respect to their left elements, while successive pairs with the same right element can be ordered arbitrarily, respectively).

▶ Remark 3. If an order $\preceq'$ on $V_{\mathcal{D}}$ is explicitly given as a bijection $\pi : V_{\mathcal{D}} \to \{1, \ldots, n\}$, then we can modify $\mathcal{D}$ (in $\mathrm{O}(|V_{\mathcal{D}}|)$) such that $\preceq_{\mathcal{D}} = \preceq'$. In this regard, sorted RPQ-Enum just models the case where we wish the enumeration to be sorted according to some order. In particular, by assuming the order $\preceq_{\mathcal{D}}$ to be implicitly represented by $\mathcal{D}$, we *do not* hide the non-linear complexity of sorting n numbers (this would only be the case if we would generally assume that $V_{\mathcal{D}} \subseteq \mathbb{N}$ and then require sorted enumeration with respect to $\leq$).

2.3 General Algorithmic Framework for RPQ Evaluation

We assume the RAM model with logarithmic word-size as our computational model. Let us next discuss our algorithmic framework for RPQ evaluation. The input to our algorithms is a graph database $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ and an RPQ q (and, for solving the problem RPQ-Test, also a pair $(u, v) \in V_{\mathcal{D}}$).

In the case of RPQ-Enum, the algorithms have routines preprocess and enum. Initially, preprocess performs some preliminary computations on the input or constructs some auxiliary data structures; the performance of preprocess is measured in its running time depending on the input size as usual (i. e., we treat preprocess as an individual algorithm). Then enum will produce an enumeration $(u_1, v_1), (u_2, v_2), \ldots, (u_\ell, v_\ell)$ such that $q(\mathcal{D}) = \{(u_i, v_i) \mid 1 \leq i \leq \ell\}$, no element occurs twice, and the algorithm reports when the enumeration is done. We measure the performance of enum in terms of its *delay*, which describes the time that (in the worst-case) elapses between enumerating two consecutive elements, between the start of the enumeration and the first element, and between the last element and the end of the enumeration (or between start and end in case that $q(\mathcal{D}) = \emptyset$). We say that (variants of) RPQ-Enum can be solved *with preprocessing* p *and delay* d, where p and d are functions bounding the preprocessing running time and the delay. In the case that $p = \mathrm{O}(d)$, the preprocessing complexity is absorbed by the delay; in this case, we say that (variants of) RPQ-Enum can be solved *with delay* d and do not mention any bound on the preprocessing.

We also consider RPQ-Enum in the *dynamic setting*, i. e., there is the possibility to perform *update* operations on the input graph database $\mathcal{D}$, which triggers a routine update. After an update and termination of the update routine, invoking enum is supposed to enumerate $q(\mathcal{D}')$, where $\mathcal{D}'$ is the updated graph database. The performance of an algorithm for RPQ-Enum is then measured in the running times of routines preprocess (to be initially carried out only once) and update, as well as the delay. We only consider the following types of individual updates: inserting a new arc between existing nodes, deleting an arc, adding a new (isolated) node, deleting an (isolated) node. In particular, deleting or adding a single non-isolated node u may require a non-constant number of updates.

3 The Product Graph Approach

The PG-approach has already been informally described in the introduction; for our fine-grained perspective, we need to define it in detail. Let $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$ be a graph database over some alphabet Σ and let q be an RPQ over Σ. Furthermore, let (G_q, p_0, p_f) with $G_q = (V_q, E_q)$ be an automaton for q. Recall that, according to Proposition 2, G_q can be obtained in time $\mathrm{O}(|q|)$ and it has $\mathrm{O}(|q|)$ states and $\mathrm{O}(|q|)$ arcs. The *product graph* of $\mathcal{D}$ and G_q is the Σ-graph $G_{\boxtimes}(\mathcal{D}, q) = (V_{\boxtimes}(\mathcal{D}, q), E_{\boxtimes}(\mathcal{D}, q))$, where $V_{\boxtimes}(\mathcal{D}, q) = \{(u, p) \mid u \in V_{\mathcal{D}}, p \in V_q\}$ and

$$\begin{aligned} E_{\boxtimes}(\mathcal{D}, q) = \{&((u,p), x, (v,p')) \mid (u,x,v) \in E_{\mathcal{D}}, (p,x,p') \in E_q\} \cup \\ \{&((u,p), \varepsilon, (u,p')) \mid u \in V_{\mathcal{D}}, (p,\varepsilon,p') \in E_q\}\,. \end{aligned}$$

▶ Remark 4. The arc labels in $G_{\boxtimes}(\mathcal{D}, q)$ are superfluous in the sense that we only need the underlying graph of $G_{\boxtimes}(\mathcal{D}, q)$ (see Lemma 6). We define it nevertheless as Σ-graph, since then all our definitions and terminology for Σ-graphs introduced above apply as well.

▶ **Lemma 5.** $|V_{\boxtimes}(\mathcal{D}, q)| = \mathrm{O}(|V_{\mathcal{D}}||q|)$, $|E_{\boxtimes}(\mathcal{D}, q)| = \mathrm{O}(|\,\mathcal{D}\,||q|)$ *and* $G_{\boxtimes}(\mathcal{D}, q)$ *can be computed in time* $\mathrm{O}(|G_{\boxtimes}(\mathcal{D}, q)|) = \mathrm{O}(|\,\mathcal{D}\,||q|)$.

The following lemma, which is an immediate consequence of the construction, shows how $G_{\boxtimes}(\mathcal{D}, q)$ can be used for solving RPQ evaluation tasks (recall that E^* is the reflexive-transitive closure of the underlying unlabelled graph).

▶ **Lemma 6.** *For every* $u, v \in V_{\mathcal{D}}$, $(u, v) \in q(\mathcal{D})$ *if and only if* $((u, p_0), (v, p_f)) \in (E_{\boxtimes}(\mathcal{D}, q))^*$.

4 Fine-grained Complexity and Conditional Lower Bounds

We now state several computational problems along with hypotheses regarding their complexity, which are commonly used in the framework of fine-grained complexity to obtain conditional lower bounds. We discuss some details and give background information later on.

- Orthogonal Vectors (OV): Given sets A, B each containing n Boolean-vectors of dimension d, check whether there are vectors $\vec{a} \in A$ and $\vec{b} \in B$ that are orthogonal.
  OV-Hypothesis: For every $\epsilon > 0$ there is no algorithm solving OV in time $\mathrm{O}(n^{2-\epsilon}\,\mathsf{poly}(d))$.
- Boolean Matrix Multiplication (BMM): Given Boolean $n \times n$ matrices A, B, compute $A \times B$.
  com-BMM-Hypothesis: For every $\epsilon > 0$ there is no combinatorial algorithm that solves BMM in time $\mathrm{O}(n^{3-\epsilon})$.
- Sparse Boolean Matrix Multiplication (SBMM): Like BMM, but all matrices are represented as sets $\{(i,j) \mid A[i,j] = 1\}$ of 1-entries.
  SBMM-Hypothesis: There is no algorithm that solves SBMM in time $\mathrm{O}(m)$ (where m is the total number of 1-entries, i. e., $m = |\{(i,j) \mid A[i,j] = 1\}| + |\{(i,j) \mid B[i,j] = 1\}| + |\{(i,j) \mid (A \times B)[i,j] = 1\}|$).
- Online Matrix-Vector Multiplication (OMv): Given Boolean $n \times n$-matrix M and a sequence $\vec{v}^1, \vec{v}^2, \dots, \vec{v}^n$ of n-dimensional Boolean vectors, compute sequence $M\vec{v}^1, M\vec{v}^2, \dots, M\vec{v}^n$, where $M\vec{v}^i$ is produced as output before $\vec{v}^{i+1}$ is received as input.
  OMv-Hypothesis: For every $\epsilon > 0$ there is no algorithm that solves OMv in time $\mathrm{O}(n^{3-\epsilon})$.

We will reduce these problems to variants of RPQ evaluation problems in such a way that algorithms with certain running times for RPQ evaluation would break the corresponding hypotheses mentioned above. Thus, we obtain lower bounds for RPQ enumeration that are conditional to these hypotheses. In the following, we give a very brief overview of the relevance of these problems and corresponding hypotheses in fine-grained complexity.

The problem OV can be solved by brute-force in time $\mathrm{O}(n^2 d)$ and the hypothesis that there is no subquadratic algorithm is well-established. It exists in slightly different variants and has been formulated in several different places in the literature (e. g., [21, 22, 50]). The variant used here is sometimes referred to as *moderate dimension* OV-hypothesis in contrast to *low dimension* variants, where d can be assumed to be rather small in comparison to n. The relevance of the OV-hypothesis is due to the fact that it is implied by the Strong Exponential Time Hypothesis (SETH) [46, 47], and therefore it is a convenient tool to prove SETH lower bounds that has been applied in various contexts.

One of the most famous computational problems is BMM, which, unfortunately, is a much less suitable basis for conditional lower bounds. The straightforward algorithm solves it in time $\mathrm{O}(n^3)$, but there are *fast matrix multiplication* algorithms that run in time $\mathrm{O}(n^{2.373})$ [49, 30]. It is unclear how much further this exponent can be decreased and there is even belief that BMM can be solved in time $n^{2+\mathrm{o}(1)}$ (see [48] and Section 6 of [13]). However, these theoretically fast algorithms cannot be considered efficient in a practical sense, which motivates the mathematically informal notion of "*combinatorial*" algorithms (see, e. g., [52]).[1] So far, no truly subcubic *combinatorial* BMM-algorithm exists and it has been shown in [52] that BMM is contained in a class of problems (including other prominent examples like Triangle Finding (also mentioned below) and Context-Free Grammar Parsing) which are all equivalent in the sense that if one such problem is solvable in truly subcubic time by a combinatorial algorithm, then all of them are. Consequently, it is often possible to argue that the existence of a certain combinatorial algorithm for some problem would imply a major (and unlikely) algorithmic breakthrough with respect to BMM, Parsing, Triangle Finding, etc. Despite the defect of relying on the vague notion of *combinatorial* algorithms, this lower bound technique is a common approach in fine-grained complexity (see, e. g., [52, 32, 2, 3, 1, 33]). Whenever we use the com-BMM-hypothesis, our reductions will always be combinatorial, which is necessary; moreover, whenever we say that a certain running time cannot be achieved unless the com-BMM-hypothesis fails, we mean, of course, that it cannot be achieved by a combinatorial algorithm.

In order to make BMM suitable as base problem for conditional lower bounds (that does not rely on *combinatorial* algorithms) one can formulate the weaker (i. e., more plausible) hypothesis that BMM cannot be solved in time linear in the number of 1-entries of the matrices (therefore called *sparse* BMM since matrices are represented in a sparse way); see [4, 54]. Another approach is to require the output matrix $A \times B$ to be computed column by column, i. e., formulating it as the online-version OMv. For OMv, subcubic algorithms are not known and would yield several major algorithmic breakthroughs (see [32]).

A convenient tool to deal with BMM is the problem Triangle: check whether a given undirected graph G has a triangle. This is due to the fact that these two problems are subcubic equivalent with respect to combinatorial algorithms (see [52]), i. e., the com-BMM-hypothesis fails if and only if Triangle can be solved by a combinatorial algorithm in time $\mathrm{O}(n^{3-\epsilon})$ for some $\epsilon > 0$. Thus, for lower bounds conditional to the com-BMM-hypothesis, we

[1] The term "combinatorial algorithm" is not well-defined, but intuitively such algorithms have running times with low constants in the O-notation, and are feasibly implementable.

can make use of both these problems. There is also a (non-combinatorial) Triangle-hypothesis that states that Triangle cannot be solved in linear time in the number of edges, but we were not able to apply it in the context of RPQ evaluation (see [2] for different variants of Triangle).

5 Bounds for the Non-Enumeration Problem Variants

We now investigate how well the PG-approach performs with respect to the non-enumeration variants of RPQ evaluation, and we give some evidence that, in most cases, it can be considered optimal or almost optimal (subject to the algorithmic hypotheses of Section 4).

5.1 Boolean Evaluation, Testing and Computing a Witness

It is relatively straightforward to see that the problems RPQ-Test and RPQ-Boole are equivalent and can both be reduced to RPQ-Witness. Hence, upper bounds for RPQ-Witness and lower bounds for RPQ-Test or RPQ-Boole automatically apply to all three problem variants, which simplifies the proofs for such bounds.

The PG-approach directly yields the following upper bound.

▶ **Theorem 7.** *RPQ-Test, RPQ-Boole and RPQ-Witness can be solved in time* $\mathrm{O}(|\mathcal{D}||q|)$.

More interestingly, we can complement this upper bound with lower bounds as follows.

▶ **Theorem 8.** *If any of the problems RPQ-Test, RPQ-Boole and RPQ-Witness can be solved in time* $\mathrm{O}(|\mathcal{D}|^{2-\epsilon} + |q|^2)$ *or* $\mathrm{O}(|\mathcal{D}|^2 + |q|^{2-\epsilon})$ *for some* $\epsilon > 0$*, then the OV-hypothesis fails. This lower bound also holds for the restriction to sparse graph databases.*

Proof Sketch. The overall reduction is similar to the reduction from [6] used for proving conditional lower bounds of regular expression matching: For an OV-instance $A = \{\vec{a}^1, \ldots, \vec{a}^n\}$ and $B = \{\vec{b}^1, \ldots, \vec{b}^n\}$, we define $q = \#(w_1 \vee w_2 \vee \ldots \vee w_n)\#$, where w_i is a bit-string representing vector $\vec{b}^i$, and we create a graph database that, for every $\vec{a}^i$, has a path whose j^{th} arc is labelled with 0 if $\vec{a}^i[j] = 1$ and with both 0 and 1 if $\vec{a}^i[j] = 0$. This yields a reduction that rules out running times of the form $\mathrm{O}((|\mathcal{D}||q|)^{1-\epsilon})$; the stronger statement of the theorem can be obtained by carefully partitioning A or B depending on ϵ into several instances and then using the above reduction (with a more careful running time analysis). ◀

Since $(|\mathcal{D}||q|)^{1-\epsilon} \leq ((\max\{|\mathcal{D}|, |q|\})^2)^{1-\epsilon} = \max\{|\mathcal{D}|^{2-2\epsilon}, |q|^{2-2\epsilon}\} \leq |\mathcal{D}|^{2-\epsilon} + |q|^2$, Theorem 8 also rules out running times of the form $\mathrm{O}((|\mathcal{D}||q|)^{1-\epsilon})$ and $\mathrm{O}(\max\{|\mathcal{D}|, |q|\}^{2-\epsilon})$, but does not exclude a running time of $\mathrm{O}(|\mathcal{D}|^{2-\epsilon} + f(|q|))$ with $f(|q|) = \Omega(|q|^2)$. However, for $\epsilon < 1$ this is super-linear in $|\mathcal{D}|$ (and therefore inferior to $\mathrm{O}(|\mathcal{D}||q|)$ under the assumption $|q| \ll |\mathcal{D}|$), and for $\epsilon = 1$, we would obtain $\mathrm{O}(|\mathcal{D}| + f(|q|))$, which, under the assumption $|q| \ll |\mathcal{D}|$, is a small and arguably negligible asymptotic improvement over $\mathrm{O}(|\mathcal{D}||q|)$.

If the size of $\mathcal{D}$ is expressed in terms of $|V_{\mathcal{D}}|$, then Theorem 7 also gives an upper bound of $\mathrm{O}(|V_{\mathcal{D}}|^2|q|)$. In this regard, we can show the following lower bound.

▶ **Theorem 9.** *If any of the problems RPQ-Test, RPQ-Boole and RPQ-Witness can be solved in time* $\mathrm{O}(|V_{\mathcal{D}}|^{3-\epsilon} + |q|^{3-\epsilon})$ *for some* $\epsilon > 0$*, then the com-BMM-hypothesis fails.*

Proof Sketch. We reduce from Triangle (see Section 4). For a given graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, we create a graph database $\mathcal{D}$ by using 4 copies $V_i = \{(u, i) \mid u \in V\}$, $i \in [4]$, of V and add all of G's arcs between each V_i and V_{i+1}, $i \in [3]$, i. e., there is an arc

$((u, i), \mathtt{a}, (v, i+1))$ if and only if $(u, v) \in E$ ($\mathtt{a}$ is a symbol). Now G contains a triangle if and only if $\mathcal{D}$ has a length-3 path from $(u, 1)$ to $(v, 4)$ with $u = v$. The problem is how to query paths that are synchronised in this way. To this end, we add a path $s_1, s_2, \ldots, s_n$ (each arc labelled with $\mathtt{a}$) and, for every $j \in [n]$, an $\mathtt{a}$-labelled arc from s_j to $(v_j, 1) \in V_1$; analogously, we add a path $t_1, t_2, \ldots, t_n$ with an arc from $(v_j, 4)$ to t_j. Now there is an $\mathtt{a}^{n+4}$-labelled path from s_1 to t_n if and only if there is a length-3 path from $(u, 1)$ to $(u, 4)$ *for some* $u \in V$. ◀

Since $\mathrm{O}((|\mathcal{D}||q|)^{1-\epsilon}) \subseteq \mathrm{O}((|V_{\mathcal{D}}|^2|q|)^{1-\epsilon}) \subseteq \mathrm{O}(|V_{\mathcal{D}}|^{3-\epsilon} + |q|^{3-\epsilon})$, such running times are also ruled out under the com-BMM-hypothesis. Especially, a combinatorial algorithm with running time $\mathrm{O}((|\mathcal{D}||q|)^{1-\epsilon})$ refutes *both* the OV- and the com-BMM-hypothesis; thus, such an algorithm does not exist provided that at least one of these hypotheses is true (basing lower bounds on several hypotheses is common in fine-grained complexity, see, e. g., [3]).

The lower bounds discussed above are only meaningful for combined complexity. However, the upper bound of Theorem 7 already yields the optimum of *linear* data complexity.

5.2 Full Evaluation and Counting

The following upper bound is again a straightforward application of the PG-approach.

▶ **Theorem 10.** *RPQ-Eval can be solved in time* $\mathrm{O}(|V_{\mathcal{D}}||\mathcal{D}||q|)$.

Instead of using graph-searching techniques on $G_{\boxtimes}(\mathcal{D}, q)$, we could also compute the complete transitive closure of $G_{\boxtimes}(\mathcal{D}, q)$ with fast matrix multiplication.

▶ **Theorem 11.** *If BMM can be solved in time* $\mathrm{O}(n^{\omega})$ *with* $\omega \geq 2$*, then RPQ-Eval can be solved in time* $\mathrm{O}(|V_{\mathcal{D}}|^{\omega}|q|^{\omega})$.

We mention this theoretical upper bound for completeness, but stress the fact that our main interest lies in *combinatorial* algorithms.

In addition to the practical limitations of fast matrix multiplication, we also observe that the approach of Theorem 11 is only better if the graph database is not too sparse, i. e., only if $|V_{\mathcal{D}}||\mathcal{D}| = \Omega(|V_{\mathcal{D}}|^{\omega})$.

Next, we investigate the question whether $\mathrm{O}(|V_{\mathcal{D}}||\mathcal{D}||q|)$ is optimal for RPQ-Eval at least with respect to combinatorial algorithms. Since for RPQ-Eval the PG-approach does not yield an algorithm that is linear in data complexity (like it was the case with respect to the problems of Section 5.1), the question arises whether the $|V_{\mathcal{D}}||\mathcal{D}|$ part can be improved at the cost of spending more time in $|q|$. It seems necessary that respective *data complexity lower bounds* need reductions that do not use q to represent a non-constant part of the instance, as it was the case for both the OV and the Triangle reduction from Section 5.1.

It is not difficult to see that the product of two $n \times n$ Boolean matrices A and B can be represented by node sets $V_i = \{(j, i) \mid j \in [n]\}$, $i \in [3]$, with arcs $((j, 1), \mathtt{a}, (k, 2))$ iff $A[j, k] = 1$, and $((k, 2), \mathtt{a}, (\ell, 3))$ iff $B[k, \ell] = 1$, and the RPQ $q = \mathtt{aa}$, i. e., $((i, 1), (j, 3)) \in q(\mathcal{D})$ iff $(A \times B)[i, j] = 1$ (it is, of course, crucial that q has constant size). This shows that, for combinatorial algorithms and subject to the com-BMM-hypothesis, $\mathrm{O}(|V_{\mathcal{D}}||\mathcal{D}|)$ is a tight bound for the data complexity of RPQ-Eval.

▶ **Theorem 12.** *If RPQ-Eval can be solved in time* $\mathrm{O}((|V_{\mathcal{D}}||\mathcal{D}|)^{1-\epsilon} f(|q|))$ *for some function* f *and some* $\epsilon > 0$*, then the com-BMM-hypothesis fails.*

If we drop the restriction to combinatorial algorithms, we can nevertheless show (with more or less the same construction) that linear time in data complexity is impossible, unless the SBMM-hypothesis fails. However, since the size of the output $q(\mathcal{D})$ might be super-linear in $|\mathcal{D}|$, we should interpret *linear* as linear in $|\mathcal{D}| + |q(\mathcal{D})|$.

▶ **Theorem 13.** *If RPQ-Eval can be solved in time* $\mathrm{O}((|q(\mathcal{D})| + |\mathcal{D}|)f(|q|))$ *for some function* f*, then the SBMM-hypothesis fails.*

Surprisingly, we can obtain a more complete picture for the problem RPQ-Count. First, we observe that obviously all upper bounds carry over from RPQ-Eval to RPQ-Count. On the other hand, a combinatorial $\mathrm{O}((|V_{\mathcal{D}}||\mathcal{D}|)^{1-\epsilon}f(q))$ algorithm or a general $\mathrm{O}((|q(\mathcal{D})| + |\mathcal{D}|)f(|q|))$ algorithm for RPQ-Count does not seem to help for solving Boolean matrix multiplication (and therefore, the lower bounds do not carry over). Fortunately, it turns out that OV is a suitable problem to reduce to RPQ-Count, although by a rather different reduction compared to the one used for Theorem 8.

▶ **Theorem 14.** *If RPQ-Count can be solved in time* $\mathrm{O}(|\mathcal{D}|^{2-\epsilon}f(|q|))$ *for some function* f *and* $\epsilon > 0$*, then the OV-hypothesis fails.*

Proof Sketch. Let $A = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n\}$ and $B = \{\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n\}$ be an OV-instance, and let A' be the Boolean matrix having rows $\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_n$ and let B' be the Boolean matrix having columns $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_n$. Then $(A' \times B')[i,j] = 0$ if and only if $\vec{a}_i$ and $\vec{b}_j$ are orthogonal. If we represent $A' \times B'$ as a graph database and a size-2 RPQ q (as sketched above), we have that $|q(\mathcal{D})|$ is the number of 1-entries in $A' \times B'$ and therefore we can solve the OV-instance (A, B) by solving RPQ-Count for $\mathcal{D}$ and q. ◀

Since Theorem 14 also excludes running time $\mathrm{O}((|V_{\mathcal{D}}||\mathcal{D}|)^{1-\epsilon}f(|q|))$ for any function f and $\epsilon > 0$ (without restriction to combinatorial algorithms), it also shows that, subject to the OV-hypothesis, $\mathrm{O}(|V_{\mathcal{D}}||\mathcal{D}|)$ is a tight bound for the data complexity of RPQ-Count.

6 Bounds for the Enumeration of RPQs

By using the PG-approach for enumeration, we can obtain the following upper bound.

▶ **Theorem 15.** *Sorted RPQ-Enum can be solved with* $\mathrm{O}(|\mathcal{D}||q|)$ *delay and* $\mathrm{O}(1)$ *updates.*

Proof Sketch. We store all nodes (u, p_0) of $G_{\boxtimes}(\mathcal{D}, q)$ in an array S, all nodes (v, p_f) in an array T, and the general idea is to perform an individual BFS from each node in S. However, in order to bound the delay, we first have to determine the subset $S' \subseteq S$ of nodes that can reach at least one node from T. To this end, we add a new node v_{sink} with incoming arcs from each node in T, and then we perform a BFS from v_{sink} in reverse direction. This concludes the preprocessing (and, with a bit of care, this can be done in time $\mathrm{O}(|G_{\boxtimes}(\mathcal{D}, q)|) = \mathrm{O}(|\mathcal{D}||q|)$). In the enumeration, we perform a BFS from each $(u, p_0) \in S'$ and we consider these start nodes ordered by $\preceq$. In each individual BFS, we first collect in an array all visited nodes $(v_1, p_f), (v_2, p_f), \ldots$ and then enumerate all pairs (u, v_i) sorted by their right element according to $\preceq$. Due to our preprocessing, we produce at least one pair in each BFS, and each BFS can be done in time $\mathrm{O}(|G_{\boxtimes}(\mathcal{D}, q)|) = \mathrm{O}(|\mathcal{D}||q|)$, including the time needed for enumerating the pairs. Updates can be maintained trivially, since the delay bound allows to repeat the preprocessing with respect to the updated database. ◀

This enumeration algorithm is easy to implement and has some nice features like linear preprocessing (in data complexity), sorted enumeration and constant updates. Unfortunately, these features come more or less for free with the disappointing delay bound. Is the PG-approach therefore the wrong tool for RPQ-enumeration? Or can we give evidence that linear delay is a barrier we cannot break? The rest of this work is devoted to this question.

Since running an algorithm for RPQ-Enum until we get the first element yields an algorithm for RPQ-Boole (with preprocessing plus delay as running time), and since running such an algorithm completely solves RPQ-Eval in time preprocessing plus $|q(\mathcal{D})|$ times delay, we can inherit several lower bounds directly from Section 5.

▶ **Theorem 16.** *If, for some function f and $\epsilon \geq 0$, RPQ-Enum can be solved with*

1. *delay $\mathrm{O}(|\mathcal{D}|^{2-\epsilon} + |q|^2)$ or $\mathrm{O}(|\mathcal{D}|^2 + |q|^{2-\epsilon})$, then the OV-conjecture fails.*
2. *delay $\mathrm{O}(|V_{\mathcal{D}}|^{3-\epsilon} + |q|^{3-\epsilon})$, then the com-BMM-hypothesis fails.*
3. *preprocessing $\mathrm{O}(|\mathcal{D}|f(|q|))$ and delay $\mathrm{O}(f(|q|))$, then the SBMM-hypothesis fails.*
4. *prep. $\mathrm{O}(|V_{\mathcal{D}}|^{3-\epsilon}f(|q|))$ and delay $\mathrm{O}(|V_{\mathcal{D}}|^{1-\epsilon}f(|q|))$, then the com-BMM-hypothesis fails.*

The first two bounds only tell us that we might not be able to lower the delay $\mathrm{O}(|\mathcal{D}||q|)$ in terms of combined complexity. While this point of view was justified for the problems discussed in Section 5, it does not say anything regarding delays of the form $\mathrm{O}(|\mathcal{D}|^{1-\epsilon}f(|q|))$. The third bound, conditional to the SBMM-hypothesis, is much more relevant, since it suggests that the optimum of linear preprocessing and constant delay is not reachable. For combinatorial algorithms, the fourth bound at least answers our main question with $|\mathcal{D}|$ replaced by $|V_{\mathcal{D}}|$: with linear preprocessing, we cannot get below a delay of $\mathrm{O}(|V_{\mathcal{D}}|)$.

These lower bounds can be improved significantly, if we also want to handle updates (within some reasonable time bounds).

▶ **Theorem 17.** *If RPQ-Enum can be solved with*

1. *arbitrary preprocessing, $\mathrm{O}(|V_{\mathcal{D}}|^{1-\epsilon}f(|q|))$ updates and $\mathrm{O}(|V_{\mathcal{D}}|^{1-\epsilon}f(|q|))$ delay for some function f and $\epsilon \geq 0$, then the OMv-hypothesis fails.*
2. *$\mathrm{O}(|V_{\mathcal{D}}|^{3-\epsilon}f(|q|))$ preprocessing, $\mathrm{O}(|V_{\mathcal{D}}|^{2-\epsilon}f(|q|))$ updates and $\mathrm{O}(|V_{\mathcal{D}}|^{2-\epsilon}f(|q|))$ delay for some function f and $\epsilon \geq 0$, then the com-BMM-hypothesis fails.*

Proof Sketch. OMv-reduction: Let M, $\vec{v}^1, \vec{v}^2, \ldots, \vec{v}^n$ be an OMv instance. We set $q = \mathtt{aa}$ and start with an empty graph database, which, by $\mathrm{O}(n^2)$ updates, is turned into $\mathcal{D}_{M,\vec{v}^1} = (\{u_i, v_i, w \mid 1 \leq i \leq n\}, \{(u_i, \mathtt{a}, v_j) \mid M[i,j] = 1\} \cup \{(v_j, \mathtt{a}, w) \mid \vec{v}^1[j] = 1\})$. Enumerating $q(\mathcal{D}_{M,\vec{v}^1})$ gives $\{(u_i, w) \mid (M\vec{v}^1)[i] = 1\}$ which represent $M\vec{v}^1$. Analogously, we produce $M\vec{v}^i$ with $q(\mathcal{D}_{M,\vec{v}^i})$ for each $2 \leq i \leq n$, where $\mathcal{D}_{M,\vec{v}^i}$ can be obtained from $\mathcal{D}_{M,\vec{v}^{i-1}}$ by n updates.
Triangle-reduction: The idea is similar to the reduction of Theorem 9. We transform a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ into a graph database $\mathcal{D}_{v_1}$ with 4 copies $V_i = \{(u, i) \mid u \in V\}$, $i \in [4]$, of V and an arc $((u,i), \mathtt{a}, (v, i+1))$ if and only if $(u, v) \in E$. Instead of appending length n-paths at V_1 and V_4 (as done for Theorem 9), we only add arcs $(s, \mathtt{a}, (v_1, 1))$, $((v_1, 4), \mathtt{a}, t)$ and set $q = \mathtt{aaaaa}$. Now $q(\mathcal{D}_{v_1}) = \{(s,t)\}$ if there is a triangle that contains v_1, and $q(\mathcal{D}_{v_1}) = \emptyset$ otherwise. For every $2 \leq i \leq n$, we can obtain $\mathcal{D}_{v_i}$ from $\mathcal{D}_{v_{i-1}}$ by performing a constant number of updates, and then enumerate $q(\mathcal{D}_{v_i})$. ◀

The first bound rules out that we can get below $\mathrm{O}(|V_{\mathcal{D}}|)$ for delay *and* update time, regardless of the preprocessing; the second one analogously rules out anything below $\mathrm{O}(|\mathcal{D}|)$ (for combinatorial algorithms and linear preprocessing). While all these lower bounds suggest that improving on the linear delay may be rather difficult, they leave the following case open.

▶ **Question 18.** *Can RPQ-Enum be solved with $\mathrm{O}(|\mathcal{D}|)$ preprocessing and $\mathrm{O}(|V_{\mathcal{D}}|)$ delay in data complexity?*

What is a reasonable conjecture with respect to this question? A combinatorial algorithm that answers it in the positive does not seem to have any unlikely consequences. Indeed, it would just entail an $\mathrm{O}(|\mathcal{D}||V_{\mathcal{D}}|)$ algorithm for RPQ-Eval (which exactly fits to Theorem 10), an $\mathrm{O}(|V||G|)$ algorithm for computing transitive closures and an $O(n^3)$ algorithm for multiplying Boolean $n \times n$ matrices, which is the state of the art for combinatorial algorithms (this is due to the obvious reductions from these problems to RPQ-Enum). However, since Question 18 is about enumeration, a positive answer also means that after $\mathrm{O}(|G|)$ preprocessing, we can enumerate the transitive closure of a graph with delay $\mathrm{O}(|V|)$, and that after $\mathrm{O}(n^2)$ preprocessing, all 1-entries of the Boolean matrix multiplication can be enumerated with delay $\mathrm{O}(n)$. Are such enumeration algorithms unlikely, so that we should rather expect a negative answer to Question 18? In fact not, since for the simple RPQs $q = \mathsf{a}^*$ or $q = \mathsf{aa}$, which are sufficient to encode transitive closures and Boolean matrix multiplications (see also the sketch on page 11), linear preprocessing and delay $\mathrm{O}(|V_{\mathcal{D}}|)$ is indeed possible, as we shall obtain as byproducts of the results in the next section.

We close this section by the following remark that points out some similarity (and differences) of reductions used in Sections 5 and 6.

▶ Remark 19. As already mentioned in Section 5.1, the reduction used for the combined complexity lower bounds of Theorems 8 (and therefore Point 1 of Theorem 16) is similar to the reduction from [6] used for proving conditional lower bounds of regular expression matching. Moreover, the OV-lower bound for RPQ-Count of Theorem 14 is similiar to a lower bound on counting the results of certain conjunctive queries from [14], and the OMv-lower bound from Point 1 of Theorem 17 is similiar to a lower bound on enumerating certain conjunctive queries from [14].

The quite simple observation that Boolean matrix multiplication can be expressed as querying a bipartite graph (with conjunctive queries) has also been used in [8] (see also [13, Section 6]) and is also the base for the OMv-lower bound of [14]. In the context of this paper, this connection has been used in the bounds of Theorems 12, 13 and Points 3 and 4 of Theorem 16.

The obvious connection between evaluating (non-acyclic) conjunctive queries and finding triangles (or larger cliques) has already been observed in [20] (see also [13, Section 6]). However, the Triangle-lower bounds of this paper are quite different, since RPQs cannot explicitly express the structure of a triangle (or a larger clique, for that matter) by using conjunction. Therefore, our respective lower bounds (Theorems 9, Point 2 of Theorem 16, and Point 2 of Theorem 17) need to encode this aspect in a different way. With respect to Theorems 9 and Point 2 of Theorem 16 this is done by using non-constant queries (which explains why the lower bounds are not for data complexity, in contrast to the case of conjunctive queries), and with respect to Point 2 of Theorem 17, which *does* work for data complexity, it is done by using updates. Moreover, our Triangle-lower bounds do not seem to extend to larger cliques like it is the case for conjunctive queries (see [13, Section 6]).

Finally, we wish to point out that although some of the reductions used in this paper are similar to reductions used in the context of conjunctive queries, due to the difference of RPQs and CQs, none of the lower bounds directly carry over. Furthermore, note that the lower bound reductions in [8, 14] have been used for obtaining dichotomies and therefore have been stated in a much more general way.

7 Enumeration with Sub-Linear Delay

We now explore three different approaches towards enumeration of $q(\mathcal{D})$ with delay strictly better than $\mathrm{O}(|\mathcal{D}|)$ (in data complexity): (1) allowing super-linear preprocessing, (2) enumerating a representative subset of $q(\mathcal{D})$, and (3) restricting the RPQs.

Regarding the first approach, we can improve the delay from $\mathrm{O}(|\mathcal{D}|)$ to $\mathrm{O}(|V_{\mathcal{D}}|)$ by increasing the linear preprocessing by a factor of $\overline{\Delta}(\mathcal{D}) \log(\overline{\Delta}(\mathcal{D}))$ (in data complexity). Recall that $\overline{\Delta}(\mathcal{D})$ is the average degree of $\mathcal{D}$.

▶ **Theorem 20.** *Sorted RPQ-Enum can be solved with* $\mathrm{O}(|q|^2 \log(\overline{\Delta}(\mathcal{D})|q|)\,\overline{\Delta}(\mathcal{D})|\,\mathcal{D}\,|)$ *preprocessing and* $\mathrm{O}(|V_{\mathcal{D}}|)$ *delay.*

Proof Sketch. The basic idea is to compute in the preprocessing for every $u \in V_{\mathcal{D}}$ a set A_u of all or at most $\overline{\Delta}(\mathcal{D})|q|$ nodes v such that $(u, v) \in q(\mathcal{D})$. With this information, we can then in the enumeration do the following for every $u \in V_{\mathcal{D}}$. If $|A_u| < \overline{\Delta}(\mathcal{D})|q|$, then A_u must contain *all* nodes reachable by a q-path from u, which we can therefore enumerate with constant delay. If $|A_u| = \overline{\Delta}(\mathcal{D})|q|$, then, by producing every $|V_{\mathcal{D}}|$ steps a pair (u, v) with $v \in A_u$, we can afford to run a whole BFS on the product graph without exceeding the delay of $|V_{\mathcal{D}}|$, which allows us to compute all remaining nodes reachable by a q-path from u.

In order to compute these sets A_u in the preprocessing, we first compute a DAG H of the strongly connected components of $G_{\boxtimes}(\mathcal{D}, q)$. Then we move through the connected components $1, 2, \ldots, \ell$ in a reverse topological sorting and compute for each component i a set B_i of at most $\overline{\Delta}(\mathcal{D})|q|$ nodes (v, p_f) (contained in earlier components $1, \ldots, i-1$) reachable from component i. This is done by initialising each B_i with the nodes (v, p_f) present in component i and then copying these sets along the backward arcs when moving through the reverse topological sorting. To avoid duplicates and to include only the smallest $\overline{\Delta}(\mathcal{D})|q|$ nodes in the sets B_i (the latter is required for sorted enumeration), we handle the sets by binary search trees with insert, delete and look-up operations in time $\mathrm{O}(\log(\overline{\Delta}(\mathcal{D})|q|))$. ◀

Obviously, in the worst case we can have $\overline{\Delta}(\mathcal{D}) = \Omega(|V_{\mathcal{D}}|)$ and then the preprocessing of the algorithm of Theorem 20 is $\Omega(|V_{\mathcal{D}}||\,\mathcal{D}\,|)$ (in data complexity) and therefore no improvement over just computing the complete set $q(\mathcal{D})$ in time $\mathrm{O}(|V_{\mathcal{D}}||\,\mathcal{D}\,|)$. However, for graph databases with low average degree, we can decrease the delay significantly from $\mathrm{O}(|\,\mathcal{D}\,|)$ to $\mathrm{O}(|V_{\mathcal{D}}|)$ at the cost of a slightly super-linear preprocessing time. As another remark about Theorem 20, we observe that the pre-computed information becomes worthless if $\mathcal{D}$ is updated. Finally, we mention a minor modification of the algorithm of Theorem 20 which yields a slightly different result that is interesting in its own right.

▶ Remark 21. At the cost of additional space (in $O(|V_{\mathcal{D}}|^2)$) and giving up on sorted enumeration, we can use the lazy array initialisation technique (see, e. g., the textbook [42]) in order to reduce the preprocessing time in Theorem 20 by a factor $\log(\overline{\Delta}(\mathcal{D})|q|)$ to $\mathrm{O}(|q|^2\,\overline{\Delta}(\mathcal{D})|\,\mathcal{D}\,|)$. More precisely, this can be achieved by implementing the sets A_u by arrays with lazy initialization instead of binary search trees.

Let us move on to the second approach. Evaluating an RPQ on a graph database $\mathcal{D}$ aims to find for each node u *all* its q-successors, i. e., nodes reachable by a q-path. It is therefore a natural restriction to ask for only *at least one* (if any) such successor. Likewise, we could also ask for at least one (if any) q-predecessor for every node. More precisely, instead of the whole set $q(\mathcal{D})$, the task is to enumerate a $q(\mathcal{D})$*-approximation*, which is a set $A \subseteq q(\mathcal{D})$ such that, for every $u, v \in V_{\mathcal{D}}$, if $(u, v) \in q(\mathcal{D})$, then also $(u, v'), (u', v) \in A$ for some $u', v' \in V_{\mathcal{D}}$. Such a set is representative for $q(\mathcal{D})$, since it contains for *every* node the information, whether it is involved as a source and whether it is involved as a target in some reachable pair from $q(\mathcal{D})$. The problem of enumerating any $q(\mathcal{D})$-approximation for given $\mathcal{D}$ and q will be denoted by App-RPQ-Enum.

▶ **Theorem 22.** *App-RPQ-Enum can be solved with linear preprocessing and constant delay.*

Proof Sketch. We store all nodes (u, p_0) of $G_{\boxtimes}(\mathcal{D}, q)$ in an array S and all nodes (v, p_f) in an array T, and we add a new node v_{source} with an arc to each $(u, p_0) \in S$. Then, we compute an array S', such that, for every $(v, p) \in V_{\boxtimes}(\mathcal{D}, q)$, if $S'[(v, p)] = 0$, then (v, p) is not reachable from any $(u, p_0) \in S$; if $S'[(v, p)] \neq 0$, then $S'[(v, p)]$ stores some $u \in V_{\mathcal{D}}$ such that

(v, p) is reachable from $(u, p_0) \in S$. To compute S' in time $\mathrm{O}(|G_{\boxtimes}(\mathcal{D}, q)|)$ we first initialise $S'[(u, p_0)] = u$ for every $(u, p_0) \in S$, and then we perform a BFS from v_{source} and always set $S'[(u', p')] = S'[(u, p)]$ for every traversed arc $((u, p), (u', p'))$. By a BFS in the opposite direction from a new node v_{sink} connected to all $(v, p_f) \in T$, we can produce an analogous array T' (storing for each node from $V_{\boxtimes}(\mathcal{D}, q)$ a reachable node from T). With S' and T', we can now enumerate a $q(\mathcal{D})$-approximation with constant delay. ◀

Interestingly, it seems rather difficult to also support updates while keeping a low delay (the following bounds are due to the same reductions used for Theorem 17, simply because these reductions produce instances for which all $q(\mathcal{D})$-approximations are equal to $q(\mathcal{D})$).

▶ **Theorem 23.** *If* App-RPQ-Enum *can be solved with*

- *arbitrary preprocessing,* $\mathrm{O}(|V_{\mathcal{D}}|^{1-\epsilon} f(|q|))$ *updates and* $\mathrm{O}(|V_{\mathcal{D}}|^{1-\epsilon} f(|q|))$ *delay, then the OMv-hypothesis fails.*
- $\mathrm{O}(|V_{\mathcal{D}}|^{3-\epsilon} f(|q|))$ *preprocessing,* $\mathrm{O}(|V_{\mathcal{D}}|^{2-\epsilon} f(|q|))$ *updates and* $\mathrm{O}(|V_{\mathcal{D}}|^{2-\epsilon} f(|q|))$ *delay for some function f and $\epsilon \geq 0$, then the com-BMM-hypothesis fails.*

Finally, as our third approach, we show that for restricted RPQ, we can solve RPQ-Enum with delay much smaller than $\mathrm{O}(|\mathcal{D}|)$. We first need some definitions. For any class $Q \subseteq$ RPQ, we denote by Enum(Q) the problem RPQ-Enum where the input RPQ is from Q. Moreover, $\bigvee(Q)$ is the set of all RPQs of the form $(q_1 \vee \ldots \vee q_m)$ with $q_i \in Q$ for every $i \in [m]$. An RPQ q over Σ is a *basic transitive* RPQ (over Σ) if $q = (x_1 \vee \ldots \vee x_k)^*$ or $q = (x_1 \vee \ldots \vee x_k)^+$, where $x_1, \ldots, x_k \in \Sigma$; and q is a *short* RPQ (over Σ) if $q = (x_1 \vee \ldots \vee x_k)$ or $q = (x_1 \vee \ldots \vee x_k)(y_1 \vee \ldots \vee y_{k'})$, where $x_1, \ldots, x_k, y_1, \ldots, y_{k'} \in \Sigma$. By BT-RPQ and S-RPQ, we denote the class of basic transitive RPQ and the class of short RPQ, respectively.

We show that for the class $\bigvee$(S-RPQ $\cup$ BT-RPQ) (which, e. g., contains RPQs of the form $q = (\mathtt{ab} \vee \mathtt{c}^* \vee \mathtt{b}(\mathtt{c} \vee \mathtt{d}) \vee (\mathtt{a} \vee \mathtt{b} \vee \mathtt{d})^+))$, semi-sorted RPQ-Enum can be solved with linear preprocessing and delay $\mathrm{O}(\Delta(\mathcal{D}))$ in data complexity (recall *semi-sorted* from Section 2).

▶ **Theorem 24.** *Semi-sorted* Enum($\bigvee$(S-RPQ $\cup$ BT-RPQ)) *can be solved with preprocessing* $\mathrm{O}(|q|^2 |\mathcal{D}|)$ *and delay* $\mathrm{O}(|q|^2 \Delta(\mathcal{D}))$.

Proof Sketch. Semi-sorted Enum(BT-RPQ): This reduces to enumerating the transitive closure of a directed graph, which can be done by starting a BFS in each $u \in V_{\mathcal{D}}$ (ordered by $\preceq$): we only have to make sure that whenever we dequeue a node v, then we first visit (and possibly enqueue) its neighbours before producing (u, v). (See also [24].)

Semi-sorted Enum(S-RPQ): Let $q = (x_1 \vee \ldots \vee x_k)(y_1 \vee \ldots \vee y_{k'})$ (the case $q = (x_1 \vee \ldots \vee x_k)$ is trivial) and note that $V_{\boxtimes}(\mathcal{D}, q)$ is partitioned into node sets V_1, V_2 and V_3 (i. e., by the right elements of the nodes, since the NFA for q has three states) and we have to enumerate all reachable pairs (u, v) from $V_1 \times V_3$. In the preprocessing, we remove all nodes from V_2 with in- or out-degree 0, and then we remove all isolated nodes from V_1 and V_3. In the enumeration, we perform a BFS from each $u \in V_1$ (ordered by $\preceq$), but we store the reached nodes from V_3 in a queue and produce (and remove from the queue) a pair whenever $\Delta(\mathcal{D})$ steps are made by the BFS after the last output. It can be shown that the queue will not run dry prematurely and therefore the delay is bounded by $\Delta(\mathcal{D})$.

Semi-sorted Enum($\bigvee$(S-RPQ $\cup$ BT-RPQ)): We show a general meta-result: if for some $Q \subseteq$ RPQ we can solve semi-sorted Enum(Q) with preprocessing p and delay d, then we can solve semi-sorted Enum($\bigvee(Q)$) with preprocessing $\mathrm{O}(|q|p)$ and delay $\mathrm{O}(|q|d)$. With the results from above, this directly implies the statement of the theorem. Let $q = (q_1 \vee \ldots \vee q_m) \in \bigvee(Q)$ with enumeration algorithms A_j for q_j. The idea is to initially perform the preprocessings for all A_j and then run their enumerations in parallel in phases $1, 2, \dots, |V_{\mathcal{D}}|$, where in phase i we get from each A_j exactly the pairs with left element i (for simplicity, assume $V_{\mathcal{D}} = [n]$ and $\preceq = \leq$). This is only possible since all A_j are semi-sorted. In phase i, we request the next elements from A_j, we discard those already produced as output and store all others as potential outputs. After we have done this, we output one of our potential outputs and then request the next elements until all A_j have finished their phase i. Since we cannot assume that we always have a new element among the m elements received next from the A_j, we have to show that we always have at least one potential output left when needed. ◀

Since $q = \mathtt{aa} \in \mathsf{S\text{-}RPQ}$ is sufficient to express BMM as RPQ evaluation (see also page 11), Theorem 24 implies that enumerating (the 1-entries of) Boolean matrix products can be solved with linear preprocessing and $\mathrm{O}(n)$ delay, but, on the other hand, this also immediately implies a matching data complexity lower bound for the upper bound of Theorem 24.

▶ **Theorem 25.** *If RPQ-Enum*($\mathsf{S\text{-}RPQ}$) *can be solved with prep.* $\mathrm{O}(|V_{\mathcal{D}}|^{3-\epsilon} f(|q|))$ *and delay* $\mathrm{O}(|\Delta(\mathcal{D})|^{1-\epsilon} f(|q|))$ *for some function f and $\epsilon \geq 0$, then the com-BMM-hypothesis fails.*

Compared to the full class of RPQs, the class $\bigvee(\mathsf{S\text{-}RPQ} \cup \mathsf{BT\text{-}RPQ})$ is quite restricted. However, comprehensive experimental analyses of query logs suggest that quite restricted RPQs are still practically relevant: in the corpus of more than 50 million RPQs analysed in [18, Table 4], roughly 50% of the RPQs are from BT-RPQ and another 25% are of the form $q = x_1 x_2 \dots x_k$, many of which with $k \leq 2$ making them S-RPQs [36].

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM J. Comput.*, 47(6):2527–2555, 2018. doi:10.1137/16M1061771.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.

3 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM J. Comput.*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.

4 Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, pages 121–126, 2009. doi:10.1145/1514894.1514909.

5 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. doi:10.1145/3104031.

6 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.

7 Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. 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 261–272, 2013.

8 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, pages 208–222, 2007.

9 Jean-François Baget, Meghyn Bienvenu, Marie-Laure Mugnier, and Michaël Thomazo. Answering conjunctive regular path queries over guarded existential rules. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 793–799, 2017. doi:10.24963/ijcai.2017/110.

10 Pablo Barceló. 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.

11 Pablo Barceló, Diego Figueira, and Miguel Romero. Boundedness of conjunctive regular path queries. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 104:1–104:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.104.

12 Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems (TODS)*, 37(4):31:1–31:46, 2012.

13 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.

14 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318, 2017.

15 Meghyn Bienvenu, Magdalena Ortiz, and Mantas Simkus. Regular path queries in lightweight description logics: Complexity and algorithms. *J. Artif. Intell. Res.*, 53:315–374, 2015. doi:10.1613/jair.4577.

16 Meghyn Bienvenu and Michaël Thomazo. On the complexity of evaluating regular path queries over linear existential rules. In *Web Reasoning and Rule Systems - 10th International Conference, RR 2016, Aberdeen, UK, September 9-11, 2016, Proceedings*, pages 1–17, 2016. doi:10.1007/978-3-319-45276-0_1.

17 Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.

18 Angela Bonifati, Wim Martens, and Thomas Timm. Navigating the maze of wikidata query logs. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 127–138, 2019. doi:10.1145/3308558.3313472.

19 Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *VLDB J.*, 29(2-3):655–679, 2020. doi:10.1007/s00778-019-00558-9.

20 Johann Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre. (The relevance of the list: propositional logic and complexity of the first order)*. PhD thesis, University of Caen Normandy, France, 2013. URL: https://tel.archives-ouvertes.fr/tel-01081392.

21 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

22 Karl Bringmann. Fine-grained complexity theory (tutorial). In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 4:1–4:7, 2019. doi:10.4230/LIPIcs.STACS.2019.4.

23 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

24 Katrin Casel, Tobias Friedrich, Stefan Neubert, and Markus L. Schmid. Shortest distances as enumeration problem. *CoRR*, abs/2005.06827, 2020. arXiv:2005.06827.

25 Katrin Casel and Markus L. Schmid. Fine-grained complexity of regular path queries. *CoRR*, abs/2101.01945, 2021. arXiv:2101.01945.

26 Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pages 323–330, 1987.

27 Diego Figueira. Containment of UC2RPQ: the hard and easy cases. In *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, pages 9:1–9:18, 2020. doi:10.4230/LIPIcs.ICDT.2020.9.

28 Diego Figueira, Adwait Godbole, Shankara Narayanan Krishna, Wim Martens, Matthias Niewerth, and Tina Trautner. Containment of simple regular path queries. *CoRR*, abs/2003.04411, 2020. arXiv:2003.04411.

29 Dominik D. Freydenberger and Nicole Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. *J. Comput. Syst. Sci.*, 79(6):892–909, 2013. doi:10.1016/j.jcss.2013.01.008.

30 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

31 Grzegorz Gluch, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. The first order truth behind undecidability of regular path queries determinacy. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, pages 15:1–15:18, 2019.

32 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30, 2015. doi:10.1145/2746539.2746609.

33 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 26:1–26:31, 2017. doi:10.4230/LIPIcs.ITCS.2017.26.

34 Leonid Libkin, Wim Martens, and Domagoj Vrgoc. Querying graphs with data. *Journal of the ACM*, 63(2):14:1–14:53, 2016.

35 Katja Losemann and Wim Martens. The complexity of regular expressions and property paths in SPARQL. *ACM Transactions on Database Systems (TODS)*, 38(4):24:1–24:39, 2013.

36 Wim Martens. Personal communication by email, October 28, 2019.

37 Wim Martens, Matthias Niewerth, and Tina Trautner. A trichotomy for regular trail queries. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, pages 7:1–7:16, 2020. doi:10.4230/LIPIcs.STACS.2020.7.

38 Wim Martens and Tina Trautner. Evaluation and enumeration problems for regular path queries. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, pages 19:1–19:21, 2018.

39 Wim Martens and Tina Trautner. Bridging theory and practice with query log analysis. *SIGMOD Rec.*, 48(1):6–13, 2019. doi:10.1145/3371316.3371319.

40 Wim Martens and Tina Trautner. Dichotomies for evaluating simple regular path queries. *ACM Trans. Database Syst.*, 44(4):16:1–16:46, 2019. doi:10.1145/3331446.

41 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing (SICOMP)*, 24(6):1235–1258, 1995.

42 Bernard M. E. Moret and Henry D. Shapiro. *Algorithms from P to NP (Vol. 1): Design and Efficiency.* Benjamin-Cummings Publishing Co., Inc., USA, 1991.

43 Juan L. Reutter, Miguel Romero, and Moshe Y. Vardi. Regular queries on graph databases. *Theory of Computing Systems (ToCS)*, 61(1):31–83, 2017.

44 Miguel Romero, Pablo Barceló, and Moshe Y. Vardi. The homomorphism problem for regular graph patterns. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005106.

45 Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Record*, 44(1):10–17, 2015.

46 Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, pages 1227–1237, 2004. doi:10.1007/978-3-540-27836-8_101.

47 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

48 Virginia Vassilevska Williams. Algorithms column: An overview of the recent progress on matrix multiplication. *SIGACT News*, 43(4):57–59, 2012. doi:10.1145/2421119.2421134.

49 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898. ACM, 2012. doi:10.1145/2213977.2214056.

50 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 17–29, 2015. doi:10.4230/LIPIcs.IPEC.2015.17.

51 Virginia Vassilevska Williams. Some open problems in fine-grained complexity. *SIGACT News*, 49(4):29–35, 2018. doi:10.1145/3300150.3300158.

52 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

53 Peter T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1):50–60, 2012. doi:10.1145/2206869.2206879.

54 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.