Dichotomies for Evaluating Simple Regular Path Queries

WIM MARTENS, University of Bayreuth, Germany
TINA TRAUTNER, University of Bayreuth, Germany

Regular path queries (RPQs) are a central component of graph databases. We investigate decision and enumeration problems concerning the evaluation of RPQs under several semantics that have recently been considered: arbitrary paths, shortest paths, paths without node repetitions (simple paths), and paths without edge repetitions (trails).

Whereas arbitrary and shortest paths can be dealt with efficiently, simple paths and trails become computationally difficult already for very small RPQs. We study RPQ evaluation for simple paths and trails from a parameterized complexity perspective and define a class of *simple transitive expressions* that is prominent in practice and for which we can prove dichotomies for the evaluation problem. We observe that, even though simple path and trail semantics are intractable for RPQs in general, they are feasible for the vast majority of RPQs that are used in practice. At the heart of this study is a result of independent interest: the two disjoint paths problem in directed graphs is W[1]-hard if parameterized by the length of one of the two paths.

CCS Concepts: • **Information systems → Query languages for non-relational engines**; • **Theory of computation → Database query languages (principles)**; *Regular languages.*

Additional Key Words and Phrases: Graph databases, regular path queries, regular languages, parameterized complexity

ACM Reference Format:
Wim Martens and Tina Trautner. ????. Dichotomies for Evaluating Simple Regular Path Queries. *ACM Trans. Datab. Syst.* ?, ?, Article ? (June ????), 45 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Regular path queries (RPQs) are an important feature of graph database query languages. They allow users to reason about complex connections in graphs by enabling them to express queries and subqueries over arbitrarily long paths. Essentially, RPQs are regular expressions that are matched against labeled directed paths in graph databases. Currently, the openCypher project [45], the LDBC Graph Query Language Task Force [3], and the World Wide Web Consortium (W3C) [54] are considering how RPQ evaluation can be formally defined for the development of Neo4j's Cypher [44, 47] and SPARQL 1.1 [53], respectively. Several popular candidates that are being considered for the semantics of RPQs are *arbitrary paths*, *shortest paths*, *simple paths*, and *trails* ([4, Section 4.4], [47]).

We briefly explain these semantics. Given a graph, an RPQ r considers directed paths for which the labels on the edges form a word in the language of r. We call such paths *candidate matches*. The different semantics restrict the kind of paths that *match* the RPQ, i.e., should be returned as answers. *Arbitrary paths* semantics imposes no restriction and returns every candidate match. *Shortest paths*

Authors' addresses: Wim Martens, University of Bayreuth, Bayreuth, Germany, wim.martens@uni-bayreuth.de; Tina Trautner, University of Bayreuth, Bayreuth, Germany, tina.trautner@uni-bayreuth.de.

0362-5915/????/6-ART? $15.00
https://doi.org/0000001.0000001

semantics, on the other hand, only returns the shortest candidate matches, *simple paths* semantics only returns candidate matches that do not have duplicate nodes, and *trails* semantics returns candidate matches that do not have duplicate edges.

Under *arbitrary paths* semantics, the number of matches may be infinite if the graph is cyclic. This may pose a challenge for designing the query language, even if one does not choose to return all matching paths. Indeed, a popular semantics of RPQs is to return *node pairs* (x, y) such that there exists a matching path from x to y. Under bag semantics for node pairs,[1] where each (x, y) is returned as often as the number of matches from x to y, one needs to deal with the case where this number is infinite.

Under *shortest paths*, *simple paths*, and *trails* semantics, the number of matching paths is always finite, which simplifies the aforementioned design challenge. However, these three versions face other challenges. *Simple paths* and *trails* semantics may present complexity issues. Two fundamental issues are that, in directed graphs, the problems of

- counting the number of simple paths or trails and
- deciding if there exists a simple path or trail of even length

from a given source to target node are hard (#P-complete [51] and NP-complete [32], respectively). Indeed, the first problem implies that evaluating the RPQ a^* under bag semantics is #P-complete and the second one implies that deciding if the RPQ $(aa)^*$ returns at least one answer is NP-complete.[2] *Shortest paths* semantics does not have these complexity issues, but it is unclear if its semantics is always natural. For instance, under shortest paths semantics, if we ask how many paths exist from x to y, then this number may decrease if a new, shorter, path is added.[3] For some queries, this behavior may seem counter-intuitive to users.

Since there may be no one-size-fits-all solution, the openCypher project team recently proposed to support several kinds of semantics for Cypher [47]. This situation motivated us to shed more light on RPQ evaluation problems, focusing on the following aspects:

- We take into account a recent study that investigated the structure of about 250K RPQs gathered from a wide range of SPARQL query logs [15]. It turns out that all these RPQs have a relatively simple structure, which is remarkable because their syntax is not restricted by the SPARQL recommendation.
- We do not only focus on decision problems but also on *enumerating the answers* to the RPQ.
- We investigate *combined complexity*, that is, problems in which the input consists of the graph G *and* the RPQ r. We do this to obtain a precise idea about the complexity of RPQ evaluation, both in terms of the data and the query.[4]

Our main message is:

> The complexity of RPQ evaluation under all four semantics (arbitrary path, shortest path, simple path, or trail) is reasonable for the types of expressions occurring in query logs. This holds both for decision versions and enumeration versions of RPQ evaluation.

More precisely, our contributions are the following:

(1) Taking into account the types of expressions occurring in the query logs of the study by Bonifati et al. [15], we define the class of *simple transitive expressions (STEs)*, which capture

[1]SPARQL 1.1 uses an approach similar to such a bag semantics.
[2]It is also known that answering the RPQ a^*ba^* under simple path semantics is at least as difficult as the Two Disjoint Paths problem [41].
[3]Notice that each semantics only returns or counts the number of paths that match.
[4]An alternative approach to the problem would be to study the so-called *data complexity*, but such an analysis considers the query to be constant, which means that the complexity in terms of the RPQ can be arbitrarily high, even for tractability results.

over 99.99% of the expressions in the logs. The remainder of the expressions are *unions of STEs*, except for one single expression.

(2) We then turn to RPQ evaluation as a decision problem. Since, in this case, RPQ evaluation for arbitrary and shortest paths is known to be tractable, we first consider simple paths. This problem is challenging because it contains special cases that are quite non-trivial. One such case is testing if there exists a directed simple path of length exactly $\log n$ between two given nodes in a graph with n nodes, which was shown to be in PTIME by Alon et al., using their color coding technique [2]. The question if it can be decided in PTIME if there is a simple path of length $\log^2 n$ has been open since 1995 [2]. Notice that these two problems are special cases of RPQ evaluation under simple path semantics (i.e., evaluate the RPQs $a^{\log n}$ and $a^{\log^2 n}$ in a graph where every edge has label a).
We therefore investigate RPQ evaluation from the angle of parameterized complexity where we use the size of the RPQ as parameter (Sections 3.5, 4.2, and 5). We identify a property of simple transitive expressions that we call *cuttability* and prove a dichotomy, showing that the parameterized complexity for evaluating a class $\mathcal{R}$ of STEs is in FPT if $\mathcal{R}$ is cuttable and W[1]-hard otherwise. Examples of cuttable classes of expressions are $\{a^k a^* \mid k \in \mathbb{N}\}$ and $\{(a+b)^k a^* \mid k \in \mathbb{N}\}$. Examples of non-cuttable classes are $\{a^k b^* \mid k \in \mathbb{N}\}$, $\{a^k b a^* \mid k \in \mathbb{N}\}$, and $\{a^k (a+b)^* \mid k \in \mathbb{N}\}$.

(3) We then turn to trail semantics and prove a dichotomy similar to the one for simple path semantics. Here we show that, if a class $\mathcal{R}$ of STEs is *almost conflict free*, the parameterized complexity of evaluation for $\mathcal{R}$ is in FPT and W[1]-hard otherwise. It should be noted that every cuttable class of expressions is also almost conflict free, which makes evaluation under trail semantics slightly "easier" than under simple path semantics.

(4) At the core of the dichotomies are two results of independent interest (Sections 4.2 and 5). The first is by the authors of [25], who showed that it can be decided in FPT if there is a simple path of length *at least* k between two nodes in a graph (Theorem 4.6). The second is proved in this article and states that the Two Disjoint Paths problem is W[1]-hard when parameterized by the length of one of the two paths (Theorem 5.5).

(5) We then turn to enumeration problems. We first observe that enumeration of arbitrary or shortest paths that match a given RPQ can be done in *polynomial delay*, i.e., such that the time between consecutive answers is polynomial (Section 8). In terms of simple paths and trails, we prove that the dichotomies on STEs carries over to the enumeration setting.

Related Work. RPQs on graph databases have been studied since the end of the 80's [18, 19, 56]. Given a graph database G, an RPQ r, and two nodes s and t, there are several natural fundamental problems associated to RPQ evaluation:

- The *decision problem*: Does r match a path from s to t in G?
- The *counting problem*: How many paths from s to t match r?
- The *computation problem*: Compute the set of paths from s to t that match r.

The decision problem is well known to be tractable for arbitrary and shortest paths by using standard automata techniques. Mendelzon and Wood [41] studied the problem for simple paths. They observed that the problem is NP-complete for a^*ba^* and $(aa)^*$. These two results heavily rely on the work of Fortune et al. [26], who showed NP-completeness of the two disjoint paths problem, and LaPaugh and Papadimitriou [32], who showed that the even length simple path problem is NP-complete.

Bagan et al. [7] provided a dichotomy for the *data complexity* of the decision problem. They defined a class C_{tract} such that the problem is in PTIME for each language in C_{tract} and NP-complete otherwise.

The *counting problem* for arbitrary paths that match an RPQ r is #P-complete in general [30].[5] However, if the RPQ is represented by a deterministic automaton (or even an unambiguous one), the counting problem is in PTIME [36], since it can be reduced to counting the number of paths in a graph *without* a restriction on the edge labels. The complexity results for arbitrary paths can easily be extended to shortest paths. Indeed, all words have equal length in Kannan et al.'s #P-hardness proof [30]. Furthermore, the PTIME algorithm for RPQs represented by deterministic or unambiguous automata also works if we need to count the words of a given length n.

The counting problem for simple paths is already #P-hard for the RPQ a^*. This immediately follows from the classical result of Valiant [51], which states that counting the number of simple paths between two given nodes in a graph is #P-complete.

Concerning the *computation problem*, Ackerman and Shallit [1] proved that one can enumerate the words accepted by a given NFA in polynomial delay. This is easily extended to RPQ evaluation w.r.t. arbitrary paths and shortest paths, as we observe in Section 8. Simple paths can be dealt with using Yen's algorithm [57], which is a method to enumerate all simple paths between two given nodes in polynomial delay. We build on this result in Section 8.2.

Yen's algorithm was generalized by Lawler [34] and Murty [43] to a tool for designing general algorithms for enumeration problems. Lawler-Murty's procedure has been used for solving enumeration problems in databases in various contexts [27, 29, 31].

Further related work concerning RPQs on graph databases are studies about the complexity of SPARQL 1.1 property paths [5, 36], which are relevant because property paths extend RPQs. The relative expressive power of graph query languages using transitive closures, data value comparisons, and branching was investigated in [35, 50]. Finally, we refer to [4, 8] for general overviews of the wide literature on graph databases.

In terms of methodology, we were heavily inspired by a line of work initiated by Frank Neven [10, 11, 39]. A practical study on the shapes of regular expressions [10] motivated the study of *simple regular expressions* and *k-occurrence regular expressions or $RE^{\leq k}$* [39] and later work on schema inference, e.g., [11].[6] Similarly, a practical study on the use of complex types in schemas for XML data [9] motivated inference algorithms for learning XML Schema [12] and the design of the BonXai schema language [38].

This article is a full version and extension of our ICDT 2018 paper [40]. In addition to providing detailed and non-trivial proofs that were absent in [40], it also includes an entirely new dichotomy for RPQ evaluation under trail semantics (Theorem 3.7).

2 PRELIMINARIES

By Σ we always denote an *alphabet*, that is, a finite set. A (Σ-)*symbol* is an element of Σ. A *word* (over Σ) is a finite sequence $w = a_1 \cdots a_n$ of Σ-symbols. The *length* of w, denoted by $|w|$, is its number of symbols n. We denote the empty word by ε. For $0 \leq i \leq j \leq n$, we denote by $w[i, j]$ the substring $a_i \cdots a_j$ of w.

We assume familiarity with regular expressions and finite automata. The regular expressions we use in this article are defined as follows: $\emptyset$, ε, and every Σ-symbol is a regular expression; and if r and s are regular expressions, then $(r \cdot s)$, $(r + s)$, and (r^*) are regular expressions. To improve readability, we use associativity and the standard priority rules to omit braces in regular expressions. We usually also omit the outermost braces. The *size* $|r|$ of a regular expression is the number of

[5] Kannan et al. proved that counting the number of words accepted by a non-deterministic automaton for a finite language is #P-complete. This result trivially extends to RPQ evaluation.

[6] Later work used the term *(extended) chain regular expressions* to refer to the simple regular expressions from [39].

occurrences of Σ-symbols in r. For example, $|((a \cdot b) \cdot a)^*| = 3$. We define the *language* $L(r)$ of r as usual.

We use the following standard abbreviations and alternative notations: (rs) abbreviates $(r \cdot s)$, $(r?)$ abbreviates $(r + \varepsilon)$, and (r^+) abbreviates (rr^*). Furthermore, if $S = \{a_1, \ldots, a_n\} \subseteq \Sigma$, then we identify S with the expression $(a_1 + \cdots + a_n)$. We allow $S = \emptyset$, in which case $L(S) = \emptyset$. As such, $L(\Sigma^*)$ contains every word and $L(\emptyset^*) = \{\varepsilon\}$. For $n \in \mathbb{N}$, we use r^n to abbreviate the n-fold concatenation $r \cdots r$ of r. We abbreviate $(r?)^n$ by $r^{\leq n}$. In the context of graph databases, *regular path queries (RPQs)* are regular expressions that can be evaluated on graphs and return an output. In this article, we will blur the distinction between them (language acceptors vs. queries) and use "regular expression" and RPQ as synonyms.

A *non-deterministic finite automaton (NFA)* N over Σ is a tuple $(Q, \Sigma, \delta, Q_I, Q_F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \times Q$ is the transition relation, $Q_I \subseteq Q$ is the set of initial states, and $Q_F \subseteq Q$ is the set of accepting states. By $\delta^*(w)$ we denote the set of states reachable by N after reading w, that is, $\delta^*(\varepsilon) = Q_I$ and, for every word w and symbol a, we define $\delta^*(wa) = \{q \mid (q', a, q) \in \delta \text{ and } q' \in \delta^*(w)\}$. The *size* of an NFA is $|Q|$, i.e., its number of states. We define the *language* $L(N)$ of N as usual.

2.1 Graph Databases

We use edge-labeled directed graphs as abstractions for graph databases. A graph G (with labels in Σ) will be denoted as $G = (V, E)$, where V is the finite set of *nodes* of G and $E \subseteq V \times \Sigma \times V$ is the set of *edges*. We say that edge $e = (u, a, v)$ goes *from node u to node v* and *has label a*. We use *a-edge* to refer to an edge with label a and *a-path* to refer to a path that consists only of a-edges. Sometimes we write an edge as $(u, v) \in V \times V$ if the label does not matter. In this article, we assume that graphs are directed, unless mentioned otherwise. Notice that our definition allows graphs to have self-loops and multi-edges. The *size* of a graph G, denoted by $|G|$, is defined as $|G| = |V| + |E|$.

A *path* from node u to node v in G is a sequence

$$p = (v_0, a_1, v_1)(v_1, a_2, v_2) \cdots (v_{n-1}, a_n, v_n)$$

of edges in G such that $u = v_0$ and $v = v_n$. For $0 \leq i \leq n$, we denote by $p[i, i]$ (or $p[i]$) the node v_i and, for $0 \leq i < j \leq n$, we denote by $p[i, j]$ the subpath $(v_i, a_{i+1}, v_{i+1}) \ldots (v_{j-1}, a_j, v_j)$. A path p is *simple* if all nodes $v_0, \ldots, v_n$ are pairwise different. It is a *trail* if it has no repeated edges, that is, all triples (v_i, a_{i+1}, v_{i+1}) are pairwise different. The *length* of p, denoted by $|p|$, is the number n of edges in p. By definition of paths, we consider two paths to be different if they are different sequences of edges. In particular, if two paths go through the same nodes in the same order and use the same edge labels, then they are the same, but if they use different edge labels, they are different. For succinctness, we sometimes also denote the path p as the sequence of nodes $v_0 v_1 \cdots v_n$ if the labels do not matter. (For instance, if we want to quantify over all such paths or if the graph does not contain two edges with different labels between the same two nodes.)

The set of *nodes of path p* is $V(p) = \{v_0, \ldots, v_n\}$. The *word of p* is $a_1 \cdots a_n$ and is denoted by $\text{lab}(p)$. Let L be a language, i.e., a set of words. Path p *matches* L if $\text{lab}(p) \in L$. If r is a regular expression (resp. N is an NFA), we simplify notation and also say that p *matches* r (resp. p *matches* N) when p matches $L(r)$ (resp., $L(N)$). The *concatenation* of paths $p_1 = (v_0, a_1, v_1) \cdots (v_{n-1}, a_n, v_n)$ and $p_2 = (v_n, a_{n+1}, v_{n+1}) \cdots (v_{n+m-1}, a_{n+m}, v_{n+m})$ is simply the concatenation $p_1 p_2$ of the two sequences. Notice that the last node of p_1 needs to be the same as the first node of p_2.

For several enumeration problems, we will consider the *radix order* on paths. To this end, we assume that there exists an order $<$ on Σ. We extend this order to words and paths. For words w_1 and w_2, we say that $w_1 < w_2$ in radix order if $|w_1| < |w_2|$ or $|w_1| = |w_2|$ and w_1 is lexicographically before w_2. For two paths p_1 and p_2, we say that $p_1 < p_2$ in radix order if $\text{lab}(p_1) < \text{lab}(p_2)$.

2.2 Enumeration Problems and Algorithms

An *enumeration problem* P is a (partial) function that maps each input i to a finite or countably infinite set of *outputs for* i, denoted by $\mathrm{P}(i)$. Terminologically, we say that, given i, the task is to *enumerate* $P(i)$.

An *enumeration algorithm* for P is an algorithm that, given input i, writes a sequence of answers to the output such that every answer in $\mathrm{P}(i)$ is written precisely once. If A is an enumeration algorithm for an enumeration problem P, we say that A runs in *polynomial delay* if the time before writing the first answer and the time between writing every two consecutive answers is polynomial in $|i|$. By *between answers*, we mean the number of steps between writing the first symbol from an answer until writing the first symbol of the next answer. We use the term *preprocessing time* to refer to the computation time before writing the first answer.

2.3 Parameterized Complexity

Several of our results will involve *parameterized complexity*, on which we give a quick overview. We follow the exposition of Cygan et al. [20] and refer to their work for further details. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$ where, as before, Σ is a fixed, finite alphabet. For an instance $(x, p) \in \Sigma^* \times \mathbb{N}$, we call p the *parameter*. The *size* $|(x, p)|$ of an instance (x, p) is defined as $|x| + p$. A parameterized problem L is called *fixed-parameter tractable* if there exists an algorithm $\mathcal{A}$, a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(x, p) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, p) \in L$ in time at most $f(p) \cdot |(x, p)|^c$. The complexity class containing exactly the fixed-parameter tractable problems is called FPT.

In terms of parameterized complexity, Downey and Fellows [22] introduced the W-hierarchy, where FPT = W[0] and $\mathrm{W}[i] \subseteq \mathrm{W}[j]$ for all $i \leq j$. It is a standard assumption in parameterized complexity theory that FPT $\neq$ W[1]. In order to prove W[1] hardness, we need the notion of *fpt-reduction.* If L and L' are two parameterized problems, an *fpt-reduction* from L to L' is an algorithm $\mathcal{R}$ that, given an instance (x, k) of L, outputs an instance (x', k') of L' such that

- (x, k) is a yes-instance of L if and only if (x', k') is a yes-instance of L',
- $k' \leq g(k)$ for some computable function g, and
- the running time of $\mathcal{R}$ is $f(k) \cdot |x|^{O(1)}$ for some computable function f.

A famous complete problem for W[1] under fpt-reductions is k-Clique with parameter k [23].

3 MAIN RESULTS

We give an overview of computational problems that we will consider in the article. All these problems are forms of the RPQ evaluation problem and their input will usually consist of two parts:

(a) a graph G, two nodes s and t in G, and
(b) an RPQ r.

As usual in database literature, part (a) is also called the *data* and part (b) the *query*. For a computational problem P and a set of RPQs $\mathcal{R}$, we denote by $\mathrm{P}(\mathcal{R})$ the problem P where the RPQ r always comes from $\mathcal{R}$. When $\mathcal{R}$ is a singleton $\{r\}$, we also write $\mathrm{P}(r)$ instead of $\mathrm{P}(\{r\})$.

If we study the *combined complexity* of a problem P, which will be the default in this article, then the input to P consists of both (a) and (b). In some cases, we will also refer to the *data complexity* of P, which means that we consider (b) to be fixed. Formally, under data complexity, each fixed RPQ r gives rise to a different computational problem $\mathrm{P}(r)$, for which the input is a graph G and nodes s and t. As such, when we say that the data complexity of a problem P has a certain upper bound, then it means that this upper bound holds for $\mathrm{P}(r)$, for every RPQ r. Likewise, when we claim a lower bound for the data complexity of P, it means that there exists an RPQ r such that $\mathrm{P}(r)$ has this lower bound.

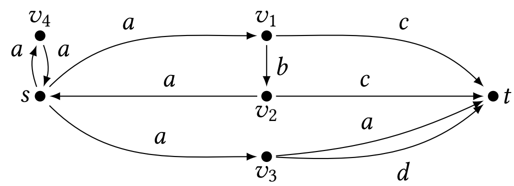

Fig. 1. A directed, edge-labeled graph

3.1 Main Problems

We now introduce the problems and the questions or the computational tasks that they ask:

(1) PathExistence: Is there a path from s to t that matches r?
(2) SimPathExistence: Is there a simple path from s to t that matches r?
(3) TrailExistence: Is there a trail from s to t that matches r?
(4) CountPaths: How many paths from s to t match r?
(5) CountShortestPaths: Among the paths from s to t that match r, how many are the shortest?
(6) CountSimplePaths: Among the paths from s to t that match r, how many are simple?
(7) CountTrails: Among the paths from s to t that match r, how many are trails?
(8) EnumShortPaths: Enumerate the shortest of the paths from s to t that match r.
(9) EnumSimPaths: Enumerate the simple paths from s to t that match r.
(10) EnumTrails: Enumerate the trails from s to t that match r.

Here, all paths are assumed to be paths in G. We consider one problem with an additional input, namely a number $\ell \in \mathbb{N}$, encoded in unary:

(11) EnumPaths: Enumerate the paths from s to t of length ℓ that match r.

The reason why we consider this extra input is because the paths can become arbitrarily large. Without the extra input ℓ, the problem would trivially not be in polynomial delay, because, from a certain point on, just writing the output cannot be done in polynomial time anymore.

Notice that each of these problems can be seen as a combination of two ingredients: a type of computational problem and a type of path. Concerning the type of computational problem, we refer to problems (1–3) as *decision problems*, problems (4–7) as *counting problems*, and problems (8–11) as *enumeration problems*. Each of these considers *arbitrary paths*, *shortest paths*, *simple paths*, or *trails*. We did not explicitly define a decision problem version for shortest paths, since this problem is the same as PathExistence.

Example 3.1. The computational problems (4-7) have the following output on the graph in Figure 1, nodes s and t, and RPQ $r = a^*b(a+d)^*$:

- CountPaths: infinite. There exists at least one matching path, at the beginning of which the a-loop from s to v_4 to s can be repeated arbitrarily often.
- CountShortestPaths: two. These paths are $(s,a,v_1)(v_1,b,v_2)(v_2,a,s)(s,a,v_3)(v_3,a,t)$ and $(s,a,v_1)(v_1,b,v_2)(v_2,a,s)(s,a,v_3)(v_3,d,t)$.
- CountSimplePaths: zero. Every path from s to t that matches r uses the node s at least twice.
- CountTrails: six. The two shortest paths, the two shortest paths prefixed with $(s,a,v_4)(v_4,a,s)$, and the two shortest paths where we add the loop $(s,a,v_4)(v_4,a,s)$ in the second visit to s, i.e., after the edge (v_2,a,s).

	Decision	Counting	Enumeration
Arbitrary paths	in PTIME [48]	in FP [48]	in polynomial delay [1]
Shortest paths	in PTIME [48]	in FP [48]	in polynomial delay [1]
Simple paths	in PTIME [48]	#P-complete [51]	in polynomial delay [48, 57]
Trails	in PTIME [48]	#P-complete [51]	in polynomial delay [48, 57]

Table 1. Complexities of fundamental path problems in graphs

	Decision	Counting	Enumeration
Arbitrary paths	in PTIME (folklore)	#P-complete [30]	in polynomial delay (Cor. 8.2)
Shortest paths	in PTIME (folklore)	#P-complete [30]	in polynomial delay (Cor. 8.2)
Simple paths	NP-complete [41]	#P-complete [30]	intractable
Trails	NP-complete	#P-complete [30]	intractable

Table 2. Complexities of fundamental RPQ evaluation problems

3.2 Complexity Background

We provide an overview of the complexities of the decision, counting, and enumeration problems when considering the different types of paths. Table 1 summarizes the complexities in the case where the RPQ does not play a role in the problems. Formally, we can do this by choosing the RPQ to be Σ^*. In this section, we will call a problem *tractable* if, assuming that $P \neq NP$, there exists a polynomial-time algorithm that produces a correct answer. In this sense, all the problems in Table 1 are tractable, except for the counting problems for simple paths and trails. The decision problems are essentially four instances of the same problem, i.e., reachability. The counting problems for arbitrary and shortest paths can be solved in FP (functional PTIME). Indeed, given a connectivity matrix A, the matrix A^k contains at entry (i, j) the number of paths of length k from node i to node j. This matrix can be computed in PTIME. Using reachability tests (or an alternative algorithm to detect loops), it can also be decided in PTIME if the number of paths from i to j is infinite. The counting problem for simple paths is one of the first problems proved to be #P-complete [51]. The problem for trails can be seen to be #P-complete by applying the standard split-graph reduction (see Lemma 6.1(2) and [46, Theorem 2.1]). Enumeration of arbitrary and shortest paths was shown to be in polynomial delay by Ackerman and Shallit [1]. Yen's algorithm is well-known to enumerate simple paths in polynomial delay [57] and can easily be adapted to work with trails using the standard reduction to directed line graphs (see our Lemma 6.2(1) and [46, Theorem 2.2]).

This table changes significantly when RPQs enter the picture. Most notably: the decision problems for simple paths and trails become NP-complete and the counting problems become #P-complete. Concerning the decision problem, the case of arbitrary and shortest paths can be solved in PTIME by finding arbitrary and shortest paths on a product between the graph database and an automaton for the RPQ (see also [41, Lemma 1]). Mendelzon and Wood [41] proved that the decision problem for simple paths is already NP-complete under data complexity [41]. More precisely, they show that the problem is NP-hard already for the expressions $(aa)^*$ and a^*ba^*. The proofs are essentially reductions from the even length simple paths problem [32] and the two disjoint paths problem [26]. The NP-hardness result for data complexity can be carried over to trails using the split-graph construction (see Lemma 6.1(1)), which splits every node in two.[7]

[7]We note that the RPQ also needs to be changed by applying this reduction. One can also adapt the reduction of Mendelzon and Wood towards trails and obtain hardness for the expressions $(aa)^*$ and a^*ba^*.

For completeness, we mention that Bagan et al. [7] investigated the data complexity of the decision problem for simple paths in much more detail and provide a trichotomy for the complexity of SimPathExistence. In particular, they define a class of RPQs C_{tract} such that SimPathExistence(r) is in PTIME for all $r \in C_{\text{tract}}$ and NP-complete otherwise.

Concerning the counting problem, Kannan et al. [30] proved that counting the number of words of a given length n in the language of an NFA N over alphabet $\{0, 1\}$ is #P-complete. The proof can trivially be adapted to produce a regular expression r_N such that $L(r_N) = L(N)$. If we consider the graph consisting of nodes $u_0, \dots, u_n$ and edges $(u_{i-1}, 0, u_i)$ and $(u_{i-1}, 1, u_i)$ for every $i = 1, \dots, n$, then the number of paths from u_0 to u_n that match the RPQ r_N is precisely the number of words of length n in $L(N)$. Therefore, the counting problem for arbitrary paths and RPQs is #P-complete. Since, on this particular graph, the answers for counting arbitrary paths, simple paths, and trails are the same, and since all paths from u_0 to u_n have the same length, all four counting problems are #P-complete.

Concerning enumeration, we show in Corollary 8.2 that enumerating arbitrary and shortest paths can be done in polynomial delay. (Essentially it can be done by a path enumeration algorithm on a product of the graph and an NFA for the RPQ.) Since already the decision problems for simple paths and trails are intractable, the enumeration problems are intractable as well.

Conclusion. From a theoretician's point of view, considering simple paths or trails for RPQ evaluation may seem computationally too complex, since already the simplest version of the decision problem is NP-complete

- under data complexity and, moreover,
- for very small RPQs such as $(aa)^*$ and a^*ba^*.

In the next section, we will see that the types of RPQs that users ask are different from those that lead to high worst-case complexity.

3.3 RPQs in Practice are Simple

Bonifati et al. [15] performed an extensive study on the structure of *property paths* in SPARQL query logs. Syntactically, SPARQL property paths are extensions of RPQs, since they have additional operators for wildcards and for following edges in the reverse direction. In Table 3, we provide a summary of the types of property paths found in the data of [15]. That is, Table 3 is not the table appearing in [15], but we went over the raw data again and aggregated the types of expressions slightly differently. In the table, we use the following conventions:

- Lower case letters denote single symbols.
- Upper case letters denote sets of symbols.
- We denote a wildcard test by $\sqcup$.[8]
- We do not distinguish between following an edge in the forward or backward direction.[9]
- Each expression type also encompasses its symmetric form. For instance, when we write a^*b, we count the expressions of the form a^*b and ba^*. We always list the variant that occurred most often in the data. That is, a^*b occurred more often than ba^*.

Under *Expression Type*, the table summarizes which types of expressions are in Bonifati et al.'s data set, sometimes parameterized by a number ℓ for which the next column describes the values that were found. *Relative* describes which percentage of the 247,404 expressions fall into this expression

[8]We treat every expression of the form $!a$ ("match every label that is not a") as a wildcard. In the total corpus, 17 expressions use the operator "!" in a slightly more complex way than just $!a$, for instance, $(!a+!b)^*$ or $(a+!a)^*$, which boil down to reachability tests in the graph and both of which we classified as $\sqcup^*$.

[9]That is, we treat the property path a the same way as $\hat{}a$. The operator $\hat{}$ was used in 306 expressions.

Expression Type	ℓ	Relative	STE?
$(a_1 + \cdots + a_\ell)^*$	2–4	29.10%	yes
$\sqcup$		25.48%	yes$^{(*)}$
a^*		19.66%	yes
$a_1 \cdots a_\ell$	2–6	8.66%	yes
a^*b		7.73%	yes
$(a_1 + \cdots + a_\ell)$	1–6	6.61%	yes
$(a_1 + \cdots + a_\ell)^+$	1–2	1.54%	yes
$a_1?a_2? \cdots a_\ell?$	1–5	1.15%	yes
$a(b_1 + b_2)?$		0.01%	yes
$a_1a_2? \cdots a_\ell?$	2–3	0.01%	yes
$a^*b?$		$< 0.01\%$	yes

Expression Type	ℓ	Relative	STE?
abc^*		$< 0.01\%$	yes
$A_1 \cdots A_\ell$	2–6	$< 0.01\%$	yes
$(a_1 + a_2)?$		$< 0.01\%$	yes
$\sqcup^*$		$< 0.01\%$	yes$^{(*)}$
$\sqcup b^*$		$< 0.01\%$	yes$^{(*)}$
$\sqcup?$		$< 0.01\%$	yes$^{(*)}$
$(ab^*) + c$		$< 0.01\%$	no
$a^* + b$		$< 0.01\%$	no
$a + b^+$		$< 0.01\%$	no
$a^+ + b^+$		$< 0.01\%$	no
$(ab)^*$		$< 0.01\%$	no

Table 3. Structure of the 247,404 SPARQL property paths that were also used in the query logs investigated by Bonifati et al. [15]. The structure is sometimes in terms of a variable $\ell \in \mathbb{N}$, for which the second column indicated the values that were found in the logs. *Relative* indicates which percentage of the 247,404 property paths have this structure.

type. We discuss *STE?* in the next section. Perhaps surprisingly, we see that the property paths found in the query logs of Bonifati et al. are not very complex.

The query logs in the study of Bonifati et al. [15] came almost exclusively from DBpedia, Semantic Web Dog Food, LinkedGeoData, BioPortal, OpenBioMed, and the British Museum, ranging from 2009 until 2016. Since the logs had 56 million unique queries, property paths did not occur often, i.e., in about 0.4% of the queries. However, this seems to be an artifact of the underlying data. In a more recent study on Wikidata query logs, containing 35 million unique queries, a drastically larger 38.94% of the queries use property paths [14]. Perhaps surprisingly, the types of expressions found in that study are similar to those presented here.

3.4 Simple Transitive Expressions

We will define *simple transitive expressions (STEs)*, with the intent of capturing the vast majority of the expressions in Table 3. Intuitively, simple transitive expressions aim at capturing the most basic navigation in graphs:

(1) first follow a path of length *exactly* k or *at most* k (for some $k \in \mathbb{N}$),
(2) then do a transitive closure step,
(3) finally, follow a path of length *exactly* ℓ or *at most* ℓ (for some $\ell \in \mathbb{N}$).

All three steps are subject to label tests. Furthermore, any step can be omitted, so a simple transitive expression can also express that paths must have length between k and $k + \ell$. Formally, we define them as follows.

Definition 3.2. An *atomic expression* is of the form $A \subseteq \Sigma$ with $A \neq \emptyset$. A *bounded expression* is a regular expression of the form $A_1 \cdots A_k$ or $A_1? \cdots A_k?$, where $k \geq 0$ and each A_i is an atomic expression. Finally, a *simple transitive expression (STE)* is a regular expression

$$B_{\text{pre}} T^* B_{\text{suff}},$$

where B_{pre} and B_{suff} are bounded expressions and T is $\emptyset$ or an atomic expression.

Notice that, by taking $T = \emptyset$, the subexpression T^* only matches ε and the STE defines a finite language. We believe that STEs capture many RPQs that users ask in practice. In Table 3 the column *STE?* indicates whether the expression is an STE. Here, we write "yes$^{(*)}$" to indicate that the expression is an STE if a wildcard is treated the same as a set of labels A. (Our algorithms indeed can be generalized to incorporate wildcards.)

In total, we saw that only 20 property paths are not STEs or trivially equivalent to an STE (by taking $T = \emptyset$ in the definition of STEs, for example). For instance, the expression type $a_1a_2? \cdots a_\ell?$ is equivalent to an STE where $B_{\text{pre}} = a_1$, $T = \emptyset$, and $B_{\text{suff}} = a_2? \cdots a_\ell?$. In this sense, 99.992% of the property paths in Table 3 correspond to STEs.

In fact, *all* expressions except for $(ab)^*$ are unions of STEs. Unions of STEs can be handled by our evaluation algorithms for simple paths and trails by running it over each STE in the union separately. The expression $(ab)^*$ is the only one left to which our techniques do not apply. It is difficult to evaluate, because even the data complexity of SimPathExistence is NP-complete for $(ab)^*$ [7]. Coincidentally, we discovered that the SPARQL query containing this expression was not generated by an ordinary user, but by a researcher who was trying to test the robustness of the SPARQL engine [52].

Proposition 3.3. *The data complexity of SimPathExistence is in polynomial time for every STE.*

Proof. The proposition states that, for every STE r, the complexity of SimPathExistence(r) is in polynomial time. This is an easy consequence of the work of Bagan et al. [7]. Following their dichotomy for the data complexity of SimPathExistence, every STE is in the class which they call $\mathrm{C}_{\text{tract}}$ and for which the problem is in polynomial time. □

3.5 Complexity Results on Simple Transitive Expressions

The main focus of the article will be a study of SimPathExistence and TrailExistence from a parameterized complexity perspective. The reason why we focus on parameterized complexity is that SimPathExistence is trivially NP-complete because it encompasses the NP-complete Hamilton Path problem. Indeed, given a graph G with n nodes and only a-edges, nodes s and t, and RPQ a^{n-1}, the SimPathExistence problem asks if there is a Hamiltonian path from s to t in G. Using Lemma 6.1(3), NP-completeness also follows for TrailExistence.

We can obtain a more precise view on the problem by looking at its parameterized complexity. Alon et al. [2] proved that SimPathExistence for graphs with n nodes and RPQs of the form a^k is fixed-parameter tractable in k, using their famous color-coding technique. We note that a precise view on the parameterized complexity of SimPathExistence subsumes long-standing open problems. For instance, SimPathExistence is in PTIME if $k = \log n$ [2], but the question if SimPathExistence is in PTIME if $k = \log^2 n$ has been open since 1995 [2].[10]

3.5.1 Two Dichotomies for Simple Transitive Expressions. We will explain our main results w.r.t. the parameterized complexity of SimPathExistence and TrailExistence, that is Theorem 3.5 and Theorem 3.7. The instances (x, p) of the problems will always be such that x encodes the graph G and regular expression r, and the parameter p is $|r|$. For every problem (1–11) from Section 3.1, we refer to its parameterized version by prefixing it with P. For instance, PSimPathExistence refers to the parameterized version of SimPathExistence.

Likewise, we denote by PSimPathExistence($\mathcal{R}$) and PTrailExistence($\mathcal{R}$) the problems PSimPath-Existence and PTrailExistence where the RPQ from the input always comes from the class of regular expressions $\mathcal{R}$. We will sometimes denote a class of RPQs by a regular expression r that uses a

[10] Björklund et al. [13] showed that, under the Exponential Time Hypothesis (ETH), for any nondecreasing polynomial time computable function f that tends to infinity there is no PTIME algorithm that can decide if there exists a simple path of length $\Omega(f(n) \log^2 n)$ between two nodes in a graph of size n. Chen and Flum [17, Theorem 12] showed that, under the ETH, deciding whether there exists a simple path of length $\log^2 n$ in an undirected graph cannot be in polynomial time.

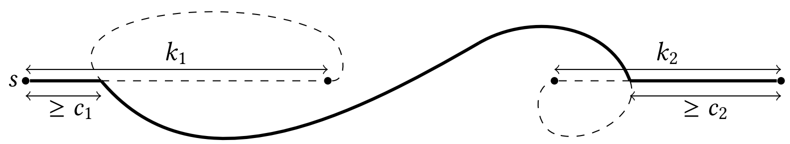

Fig. 2. Assume $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ has left and right cut borders c_1 and c_2, respectively. Assume that an arbitrary path from s to t matches r such that its length k_1 prefix and length k_2 suffix do not have loops and are node disjoint. If, after removing all loops, (1) the length c_1 prefix and length c_2 suffix are still the same and (2) the path still has length at least $k_1 + k_2$, then it matches r.

variable k. By doing so, we refer to the class of regular expressions obtained from r by replacing k by every possible number from $\mathbb{N}$. For example, $a^k b^*$ denotes the class of regular expressions $\{b^*, ab^*, aab^*, aaab^*, \ldots\}$. (We do this to be able to discuss some classes of expressions, using a simple notation. If we use this convention, we will consistently denote the variable by "k".)

Dichotomy for Simple Paths. We first define the notions that we need for the dichotomy for simple paths.

Definition 3.4. Let $r = B_{\text{pre}} T^* B_{\text{suff}}$ be an STE. If $B_{\text{pre}} = A_1 \cdots A_{k_1}$, then the *left cut border* c_1 *of* r is the largest value such that $T \not\subseteq A_{c_1}$ if it exists and zero otherwise. If $B_{\text{pre}} = A_1? \cdots A_{k_1}?$, then the left cut border is zero. Symmetrically, if $B_{\text{suff}} = A'_{k_2} \cdots A'_1$, then the *right cut border* c_2 *of* r is the largest value such that $T \not\subseteq A'_{c_2}$ if it exists and zero otherwise. (Notice that the indices in B_{suff} are reversed.) If $B_{\text{suff}} = A'_{k_2}? \cdots A'_1?$, then the right cut border is zero.

We explain the intuition behind cut borders in Figure 2. For $c \in \mathbb{N}$, an expression is *c-bordered* if the sum of its left and right cut borders is c. We call a class $\mathcal{R}$ of STEs *cuttable* if there exists a constant $c \in \mathbb{N}$ such that each expression in $\mathcal{R}$ is c'-bordered for some $c' \leq c$.

We can now prove a dichotomy on the complexity of PSimPathExistence($\mathcal{R}$) for classes of STEs $\mathcal{R}$, if $\mathcal{R}$ satisfies the following mild condition. We say that $\mathcal{R}$ *can be sampled* if there exists an algorithm that, given $k \in \mathbb{N}$, returns an expression in $\mathcal{R}$ that is k'-bordered with $k' \geq k$, and "no" if there is no such expression. We need the condition that $\mathcal{R}$ can be sampled to prove the W[1]-hardness. For this reason, this condition is no longer needed in the upper bound results (Lemma 4.17 and Theorem 8.7).

Theorem 3.5. *Let $\mathcal{R}$ be a class of STEs that can be sampled.*

(a) If $\mathcal{R}$ is cuttable, then PSimPathExistence($\mathcal{R}$) is in FPT and
(b) otherwise, PSimPathExistence($\mathcal{R}$) is W[1]-hard.

The result will follow immediately from Lemma 4.17 and Lemma 5.6. Notice that the difference between cuttable and non-cuttable classes of STEs can be subtle. For instance, $a^k b^*$ and $a^k(a+b)^*$ are non-cuttable, but $(a+b)^k a^*$ is cuttable. Looking back at Table 3, we see that abc^* is 2-bordered and all other STEs are either 0-bordered or 1-bordered. It therefore seems that cut borders in practice are small and over 99% of the expressions fall on the tractable side of Theorem 3.5.

Dichotomy for Trails. We now present a dichotomy for trails which is, perhaps surprisingly, slightly different in the sense that more classes of expressions fall on the tractable side. For instance, PTrailExistence($a^k b^*$) is in FPT because the a-path and the b-path can be evaluated independent of each other (no a-edge will be equal to a b-edge). On the other hand we have that PTrailExistence($a^k b a^*$) is W[1]-hard.

Definition 3.6. Let $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ be an STE with left cut border c_1 and right cut border c_2. We say that A_i with $i \leq c_1$ (resp., A'_j with $j \leq c_2$) is a *conflict position* if $A_i \cap T \neq \emptyset$ (resp., $A'_j \cap T \neq \emptyset$). We say that a class $\mathcal{R}$ of STEs is *almost conflict free* if there exists a constant c such that each $r \in \mathcal{R}$ has at most c conflict positions.

Observe that the class of almost conflict free STEs is larger than the class of cuttable STEs. For instance, $b^k a^3 b a^k a^*$ is almost conflict free, because every expression in the class has three conflict positions, namely the positions corresponding to the three leftmost a's. On the other hand, the left cut borders are on position $k + 4$, which can become arbitrarily large.

We say that $\mathcal{R}$ *can be conflict-sampled* if there exists an algorithm that, given $k \in \mathbb{N}$, returns an expression in $\mathcal{R}$ that has k' conflict positions with $k' \geq k$, and "no" if there is no such expression. Our main dichotomy for trails is the following.

Theorem 3.7. *Let $\mathcal{R}$ be a class of STEs that can be conflict-sampled.*
(a) If $\mathcal{R}$ is almost conflict free, then PTrailExistence($\mathcal{R}$) is in FPT and
(b) otherwise, PTrailExistence($\mathcal{R}$) is W[1]-hard.

This theorem follows immediately from Lemma 7.3 and Lemma 7.4.

3.5.2 Results for Enumeration Problems. Concerning enumeration, we prove that both Theorem 3.5(a) and Theorem 3.7(a) can be strengthened to give rise to *FPT delay* algorithms, i.e., algorithms in which the preprocessing time and delay between answers is fixed-parameter tractable. We do not revisit hardness, because already the decision versions of the problems are hard.

3.5.3 Results for Counting Problems. We do not prove new results concerning counting problems, because the picture is already relatively clear. Flum and Grohe [24, Theorem 5.1] showed that it is #W[1]-complete to count simple paths of length k in a directed graph. They do not consider dedicated source and target nodes s and t, but the problem of counting all paths of length k can easily be reduced to counting all paths of length $k + 2$ between two nodes s and t: we simply add two new nodes s and t and edges (s, v) and (v, t) for all $v \in V$. Notice that this also means that counting the number of paths of length *at least* k is hard, since the number of paths of length k is the number of paths of length at least k, minus those of length at least $k + 1$. As these hardness results don't use edge labels, the same hardness results apply for trails (using for example the reduction from Perl and Shiloach [46]). These results imply that counting is already #W[1]-hard for all classes of STEs that simply put a length constraint (length at most, at least, or exactly k) on paths, both for simple paths and trails. Notice that, for FPT results with parameter k, it does not matter if k is given in unary or binary.

3.6 What Does This Mean for Systems?

If we interpret Theorems 3.5 and 3.7 in the light of the real world property paths in Table 3 we can observe the following.

Concerning simple paths semantics, Theorem 3.5 tells us that PSimPathExistence($\mathcal{R}$) is fixed-parameter tractable for cuttable classes $\mathcal{R}$. This result, together with the observation that the largest cut border in Table 3 is two, and therefore very small, can be seen as an explanation why, in practice, simple paths semantics usually does not bring systems to their knees, even though this would theoretically be possible using regular expressions such as $(aa)^*$.

Looking closer, we prove that PSimPathExistence is in time $2^{O(|r|)} \cdot |V|^{c+3} \cdot |E|$ in the worst case (Lemma 4.17), where $|r|$ is the size of the RPQ, c is the largest cut border in $\mathcal{R}$, and $|V|$ and $|E|$ are the number of nodes and edges in the graph, respectively. In Table 3, the largest value of c in STEs or unions thereof is two (for abc^*), and $|r|$ is relatively small. We also note that this is a worst-case bound. In most practical settings, we expect that the run-time of even more naive evaluation algorithms will not come close to requiring $|V|^{c+3}$ time for these simple expressions. Indeed, a major complexity bottleneck in the evaluation algorithm is the subroutine that deals with "simple paths of length at least k", satisfying a label constraint given by the STE. For queries and graphs in which this problem is efficiently solvable, we expect STE evaluation to be efficiently possible as well.

The story for trails is similar. Here, we have that PTrailExistence($\mathcal{R}$) is fixed-parameter tractable for even more classes $\mathcal{R}$, namely those that are almost conflict-free. The precise complexity guarantees that we provide in this case are worse than for simple paths (run time $2^{O(|r|)} \cdot E^{c+6}$ in Lemma 7.3), but this is mainly because we have developed our methods for simple paths and then adapted them for trails. In this complexity bound, c does not refer to the expressions' cut border, but to its number of conflict positions, which can be smaller (but cannot be larger). Again, a major complexity bottleneck is the subroutine that deals with "trails of length at least k", with label constraints.

4 MAIN UPPER BOUND

4.1 Preliminary Technical Result: Downward Closed Languages

We first recall a useful result, Lemma 4.1, for which we need some definitions. A language is *downward closed* if it is closed under taking subsequences, that is, for every word $w = a_1 \cdots a_n \in L$ and every sequence $0 < i_1 < \cdots < i_k < n+1$, we have that $a_{i_1} \cdots a_{i_k} \in L$.[11] The *product* of graph G and NFA $N = (Q, \Sigma, \Delta, Q_I, Q_F)$ is a graph (V', E') with $V' = (V \times Q)$ and $E' = \{((u_1, q_1), a, (u_2, q_2)) \mid (u_1, a, u_2) \in E \text{ and } (q_1, a, q_2) \in \Delta\}$. We denote this product by $G \times N$. Notice that simple paths in $G \times N$ may use nodes $(u, q_1) \neq (u, q_2)$ and may therefore correspond to non-simple paths in G. We will use the following lemma to deal with downward closed parts of STEs, to be more precise, with bounded expressions of the form $A_1? \cdots A_k?$ and the transitive part T^* in the enumeration setting.

Lemma 4.1 (Theorem 5 in [41]). *Let N be an NFA for a downward closed language. Let G be a graph and s and t be nodes in G. Then we can decide if there is a simple path from s to t that matches N in time $O(|N||G|)$.*

Proof. The algorithm consists of two steps. First construct the product between N and G, which takes time $O(|N||G|)$. Then, test if (t, f) is reachable from (s, i) for some accepting state f and initial state i. Indeed, (t, f) is reachable from (s, i), if and only if there exists some path p from s to t that matches N. Since $L(N)$ is downward closed, the simple path obtained from p by removing all loops still matches N. □

Instead of reachability, we can use the algorithm of Ackerman and Shallit [1, Theorem 1] that finds a minimal word in an NFA N in $\theta(|N|^2 n^2)$ operations, where n is the length of the shortest word in $L(N)$. As a result, we can prove that, if $L(N)$ is downward closed, it is possible to output a smallest simple path in radix order that matches N in polynomial time. (If $L(N)$ is not downward closed, then the smallest path that matches N is not necessarily simple.)

Proposition 4.2. *Let N be an NFA such that $L(N)$ is downward closed. Given a graph G and two nodes s and t, a shortest simple path from s to t in G that matches N can be found in time $O(|G||N|)$ if such a path exists. A smallest such path in radix order can be found in time $O(|G|^2|N|^2|V|^2)$ if it exists.*

[11] The term *downward closed* comes from being closed under taking the smaller elements in the subsequence ordering which, due to Higman's Lemma, is a well quasi ordering.

Proof. Let G be a graph. Concerning a shortest path, the algorithm from Lemma 4.1 can easily be adjusted to return a shortest path, for instance, using breadth-first search for the reachability test. This does not influence the time bound.

Concerning a smallest path in radix order, we first observe that each shortest path from s to t in G that matches N is simple — otherwise we could obtain a shorter path by making the path simple (i.e., removing edges that form a loop), and obtain a path that still matches N because $L(N)$ is downward closed. Clearly, if i and f are an initial and accepting state of N respectively, then every shortest path from (s, i) to (t, f) in $G \times N$ corresponds (replacing nodes (u, q) with u) to a shortest path from s to t in G that matches N. Furthermore, each shortest path from s to t in G that matches N corresponds to one or more paths in $G \times N$.

So we can find a smallest simple path in radix order by viewing $G \times N$ as an NFA, using the method of Ackerman and Shallit [1, Theorem 1] to find a smallest path in radix order, and then output the corresponding path in G. We need $O(|N||G|)$ time to construct the product and $O(|N|^2|G|^2|p|^2)$ time to compute a smallest path p in radix order in $G \times N$. □

4.2 Representative Sets and Simple Paths with Length Constraints

To prove Theorem 3.5(a), we need the representative sets technique [25]. At their core, this technique can be used to prove that the following parameterized problems are in FPT:

- PSimPathLength: Given an instance (G, s, t, k) with parameter $k \in \mathbb{N}$, is there a simple path from s to t of length exactly k in G?
- PSimPathLength$^{\geq}$: Given an instance (G, s, t, k) with parameter $k \in \mathbb{N}$, is there a simple path from s to t of length at least k in G?

Before we explain the representative sets technique, we first restate some important results on these problems: Alon et al. [2] proved that PSimPathLength is in FPT, using their famous color coding technique. For the theorem statement, we assume that $G = (V, E)$.

Theorem 4.3 (Alon et al. [2]). *PSimPathLength is in time $2^{O(k)}|E| \log |V|$ and therefore in FPT.*

Bagan et al. [6, Theorem 7] combine color coding and dynamic programming to prove that, given graph G, nodes s, t, an NFA A, and a number k, deciding if there is a simple path from s to t of length at most k that matches $L(N)$ can be done in time $2^{O(k)}|N||G| \log |G|$. In their proof they actually show that it is in time $2^{O(k)}|N||G| \log |V|$. From this, the following can be inferred.

Lemma 4.4 (Immediate consequence of Corollary 1 in Bagan et al. [6]). *Let $G = (V, E)$ be a graph, s, t be nodes of G, and N be an NFA accepting a finite language. It can be decided in time $2^{O(|N|)}|G| \log |V|$ if there exists a simple path from s to t in G, labeled with a word from $L(N)$.*

Corollary 4.5. *Let $\mathcal{R}$ be a class of STEs defining finite languages. Then PSimPathExistence($\mathcal{R}$) is in FPT or, more precisely, in time $2^{O(|r|)}|G| \log |V|$.*

PSimPathLength$^{\geq}$ can be shown to be in FPT by adapting methods from Fomin et al. [25]. They proved that testing the existence of simple directed cycles of length at least k is in FPT and discovered that their technique also works for paths [21]. The following theorem is therefore by the authors of [25].

Theorem 4.6 (Similar to Theorem 5.3 in [25]). *PSimPathLength$^{\geq}$ is in FPT. More precisely, it is in time $2^{O(k)} \cdot |E||V| \log |V|$.*

We received a proof sketch of the result from Holger Dell [21] (who attributed the result to Fomin et al., the authors of [25]). Next, we provide a self-contained generalization of Theorem 4.6 that deals with edge labels, based on the proof sketch we received. Our contribution is the generalization of the approach towards the extra condition that checks the labels of the path. We emphasize that the most complex part of the proof concerns the length constraints and is due to the authors of [25].

One way to test whether there exists a simple path from s to t of length at least k is to find a simple path p_k of length exactly k such that there is a path from the last node of p_k to t that avoids p_k. But the number of such paths p_k is $n!/(k!(n-k)!)$. So naively testing and enumerating all paths is not fixed-parameter tractable in k. We therefore need a way to decrease the number of such paths we need to consider. We can do this using the following notion, originally introduced by Monien [42].

Definition 4.7 (k-representative family [25]). Given a set of nodes V, an integer $k \in \mathbb{N}$, and a set $\mathcal{S}$ containing subsets of V, all of size ℓ, for some $\ell \in \mathbb{N}$, we say that a subfamily $\hat{\mathcal{S}} \subseteq \mathcal{S}$ is *k-representative for $\mathcal{S}$* if the following holds: for every set $Y \subseteq V$ of size at most k, if there is a set $X \in \mathcal{S}$ disjoint from Y, then there is a set $\hat{X} \in \hat{\mathcal{S}}$ disjoint from Y. We abbreviate this by $\hat{\mathcal{S}} \subseteq^k_{\text{rep}} \mathcal{S}$.

Intuitively, if one needs to be able to avoid k-element sets, it is sufficient to store a k-representative set. Notice that each set $\mathcal{S}$ is trivially k-representative for itself. The crux is that we want to be able to compute k-representative sets that are small. The condition that all sets in $\mathcal{S}$ have the same size is just a technicality that allows us to simplify proofs later.

In the following, s, v are nodes and r is a regular expression of the form $A_1 \cdots A_k$ for some $k \in \mathbb{N}$. We define

$$P^r_{s,v} := \{V(p) \mid \text{ there is a simple path } p \text{ from } s \text{ to } v \text{ in } G \text{ that matches } r\}.$$

Notice that, by definition of r, these simple paths from s to v in G have length k. Therefore, all sets in $P^r_{s,v}$ have exactly $k+1$ elements.

We next show that representative sets $\hat{P}^r_{s,v} \subseteq^{k+1}_{\text{rep}} P^r_{s,v}$ exist for each node $v \in V$ and can be constructed in fixed parameter tractable time. We restate the relevant parts of Lemma 3.3 and Corollary 4.16 from [25] since we need them in the proof. Lemma 4.8 states that the relation "is a k-representative set for" is transitive. Corollary 4.9 gives a rough time and space bound for computing k-representative sets.

Lemma 4.8 (Lemma 3.3 in [25] for directed graphs). *Given a graph $G = (V, E)$ and a family $\mathcal{S}$ of subsets of V. If $\hat{\mathcal{S}} \subseteq^k_{rep} \mathcal{S}'$ and $\mathcal{S}' \subseteq^k_{rep} \mathcal{S}$, then $\hat{\mathcal{S}} \subseteq^k_{rep} \mathcal{S}$.*

Corollary 4.9 (Corollary 4.16 in [25], without weight function). *There is an algorithm that, given a family $\mathcal{A}$ of sets of size ℓ over a set V of nodes and an integer k, computes in time*

$$O\left(|\mathcal{A}| \cdot \left(\frac{k+\ell}{k}\right)^k \cdot 2^{o(k+\ell)} \cdot \log|V|\right)$$

a subfamily $\hat{\mathcal{A}} \subseteq^k_{rep} \mathcal{A}$ such that $|\hat{\mathcal{A}}| \leq \binom{k+\ell}{\ell} \cdot 2^{o(k+\ell)}$.

We now adapt Lemma 5.2 in Fomin et al. [25] to show a time and space bound for representative sets $\hat{P}^r_{s,v} \subseteq^k_{\text{rep}} P^r_{s,v}$ under label constraints. We will need this to deal with the bounded parts of STEs later.

Lemma 4.10. *For each regular expression $r = A_1 \cdots A_\ell$ and $k \geq \ell$, there is a collection of families $\hat{P}^r_{s,v} \subseteq^k_{rep} P^r_{s,v}$ with $v \in V \setminus \{s\}$, each of size at most $\binom{k+\ell+1}{\ell+1} \cdot 2^{o(k+\ell)}$. This collection of families can be computed in time $O(8^{k+o(k)}|E|\log|V| + |r||E|)$.*

Proof. Fomin et al. use in their complexity analysis that, given (u, v), one can test if there exists an edge from u to v in the graph in constant time. We first preprocess the graph so that, given $(u, v) \in V \times V$ and $i \in \{1, \ldots, \ell\}$, we can test in constant time whether there is an edge from u to

v with a label in A_i. Such preprocessing consists of annotating each edge with a ℓ-bit vector and takes time $O(|r||E|)$. (For each edge, and each A_i, test if the edge label is in A_i.)

We describe a dynamic programming algorithm. We assume w.l.o.g. that the nodes in V are named $\{s, v_1, \ldots, v_{n-1}\}$. Let D be an $\ell \times (n-1)$ matrix where the rows are indexed with integers in $1, \ldots, \ell$ and the columns are indexed with nodes in $\{v_1, \ldots, v_{n-1}\}$. For $i = 1, \ldots, \ell$, we will denote by r_i the prefix $A_1 \cdots A_i$ of r. The entry $D[i, v]$ will store a family $\hat{P}^{r_i}_{s,v} \subseteq^{k+\ell-i}_{\text{rep}} P^{r_i}_{s,v}$ of size at most $\binom{k+\ell+1}{i+1} \cdot 2^{o(k+\ell)}$. We fill the entries in the matrix D in increasing order of rows. For $i = 1$, we set $D[1, v] = \{\{s, v\}\}$ if G has an edge (s, a, v) with $a \in A_1$ and $D[1, v] = \emptyset$ otherwise. Assume that we have filled all the entries until row $i - 1$. For two families of sets $\mathcal{A}$ and $\mathcal{B}$, we define

$$\mathcal{A} \bullet \mathcal{B} = \{X \cup Y \mid X \in \mathcal{A},\ Y \in \mathcal{B},\ \text{and } X \cap Y = \emptyset\}.$$

We denote by $\exists(u, A_i, v)$ that there exists an edge (u, a, v) with $a \in A_i$. Let

$$\mathcal{N}^{r_i}_{s,v} = \bigcup_{\exists(u, A_i, v)} \hat{P}^{r_{i-1}}_{s,u} \bullet \{v\}.$$

Before we continue, we adapt Claim 5.1 in [25] such that it takes r into account, that is:

Claim 4.11. $\mathcal{N}^{r_i}_{s,v} \subseteq^{k+\ell-i}_{rep} P^{r_i}_{s,v}$

Proof. The proof is by induction on i. Let $S \in P^{r_i}_{s,v}$ and Y be a set of size at most $k + \ell - i$ such that $S \cap Y = \emptyset$. We will show that there exists a set $S' \in \mathcal{N}^{r_i}_{s,v}$ such that $S' \cap Y = \emptyset$. This will imply the desired result. Since $S \in P^{r_i}_{s,v}$, there exists a simple path $P = (s, u_1) \cdots (u_{i-1}, v)$ in G such that $S = V(P)$ and the predicate $\exists(u_{i-1}, A_i, v)$ is true. The existence of the path $P[0, i-1]$, the subpath of P from s to u_{i-1}, implies that $X' = S \setminus \{v\} \in P^{r_{i-1}}_{s,u_{i-1}}$. Take $Y' = Y \cup \{v\}$. Observe that $X' \cap Y' = \emptyset$ and $|Y'| \leq k + \ell - i + 1$. Since $\hat{P}^{r_{i-1}}_{s,u_{i-1}} \subseteq^{k+\ell-i+1}_{\text{rep}} P^{r_{i-1}}_{s,u_{i-1}}$ by induction, there exists a set $\hat{X}' \in \hat{P}^{r_{i-1}}_{s,u_{i-1}}$ such that $\hat{X}' \cap Y' = \emptyset$. However, since $\exists(u_{i-1}, A_i, v)$ and $v \notin \hat{X}'$ (because $\hat{X}' \cap Y' = \emptyset$), we have $\hat{X}' \bullet \{v\} = \hat{X}' \cup \{v\}$ and $\hat{X}' \cup \{v\} \in \mathcal{N}^{r_i}_{s,v}$. Taking $S' = \hat{X}' \cup \{v\}$ suffices for our purpose. This completes the proof of the claim. □

We fill the entry for $D[i, v]$ for $i \geq 2$ as follows. Observe that

$$\mathcal{N}^{r_i}_{s,v} = \bigcup_{\exists(u, A_i, v)} D[i-1, u] \bullet \{v\}.$$

Let us denote by $d^-(v)$ the indegree of v, i.e., the number of edges that end in v. We already have computed the family corresponding to $D[i-1, u]$ for all u. By construction, we have $|\hat{P}^{r_{i-1}}_{s,u}| \leq \binom{k+\ell+1}{i} 2^{o(k+\ell)}$ and thus also $|\mathcal{N}^{r_i}_{s,v}| \leq d^-(v) \binom{k+\ell+1}{i} 2^{o(k+\ell)}$. Furthermore, we can compute $\mathcal{N}^{r_i}_{s,v}$ in time $O\left(d^-(v)\binom{k+\ell+1}{i} 2^{o(k+\ell)}\right)$. Recall that, due to the preprocessing, we can test if there's an edge with label in A_i in constant time. Now, we use Corollary 4.9 on $\mathcal{N}^{r_i}_{s,v}$, which contains sets of size $(i+1)$, to obtain a $(k + \ell + 1 - (i+1))$-representative, i.e., $(k + \ell - i)$-representative subfamily $\hat{\mathcal{N}}^{r_i}_{s,v}$ of size at most $\binom{k+\ell+1}{i+1} \cdot 2^{o(k+\ell)}$ in time

$$O\left(d^-(v)\binom{k+\ell+1}{i} 2^{o(k+\ell)} \cdot \left(\frac{(k+\ell-i)+(i+1)}{k+\ell-i}\right)^{k+\ell-i} \cdot 2^{o((k+\ell-i)+(i+1))} \cdot \log|V|\right).$$

By Claim 4.11, we know that $\mathcal{N}^{r_i}_{s,v} \subseteq^{k+\ell-i}_{\text{rep}} P^{r_i}_{s,v}$. Thus, Lemma 4.8 implies that $\hat{\mathcal{N}}^{r_i}_{s,v} \subseteq^{k+\ell-i}_{\text{rep}} P^{r_i}_{s,v}$. We define $\hat{P}^{r_i}_{s,v} = \hat{\mathcal{N}}^{r_i}_{s,v}$ and assign this family to $D[i, v]$. This completes the description and the correctness of the algorithm.

ALGORITHM 1: FLPS ALGORITHM WITH RESTRICTED STE

Input: Graph $G = (V, E)$, nodes s, t in G, regular expression rT^* with $r = A_1 \cdots A_k$ and $T \subseteq A_i$ for all i
Output: Decide if there exists a simple path from s to t that maches rT^*

1 **for** *every* $v \in V$ **do**
2   Compute $\hat{P}^r_{s,v} \subseteq^{k+1}_{\text{rep}} P^r_{s,v}$
3   **for** *every* $X \in \hat{P}^r_{s,v}$ **do**
4         $V' \leftarrow (V \setminus X) \cup \{v\}$
5         $E' \leftarrow E \cap (V' \times T \times V')$
6   **if** *there exists a path from* v *to* t *in* (V', E') **then**
7   **return** YES
8 **return** NO

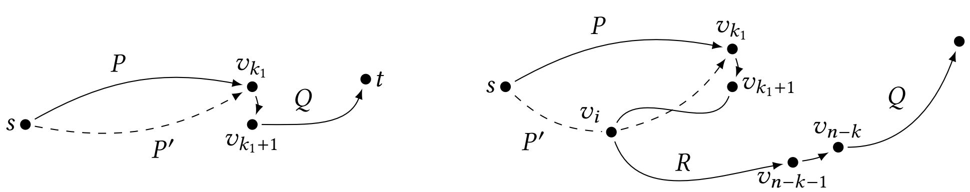

Fig. 3. This figure shows how we partition a shortest simple path p in the proof of Lemma 4.12 if p is short (left) or if p is long (right). Notice that $V(P), V(Q)$, and $V(R)$ are pairwise disjoint.

Notice that, if we keep the elements in the sets in the order in which they were built using the $\bullet$ operation, then they directly correspond to paths. As such, every ordered set in our family represents a path in the graph.

Since our only change was that we test $\exists(u, A_i, v)$ instead of the existence of an edge (u, v), the time bound $O\left(8^{k+o(k)}|E| \log |V|\right)$ [25, Lemma 5.2] carries over, modulo the additive $O(|r||E|)$ term for preprocessing that we used to test $\exists(u, A_i, v)$ in constant time. The size bound is still guaranteed by Corollary 4.9. ◻

Notice that Claim 4.11 will not work for arbitrary regular expressions. We used in the claim that if there exists an edge (u_{i-1}, a, v) with $a \in A_i$, then we can add v to any set $\hat{X}' \in \hat{P}^{r_{i-1}}_{s,u_{i-1}}$ to obtain a valid set in $\mathcal{N}^{r_i}_{s,v}$. For arbitrary regular expressions this is not the case, an example being $(aa + bb)$.

4.3 Algorithms for Simple Paths

We now present an algorithm that solves PSimPathExistence for the case where the RPQ is of the form $A_1 \cdots A_k T^*$ and is 0-bordered, that is, $T \subseteq A_i$ for all i, see Algorithm 1. The algorithm computes, for every node v, a $(k + 1)$-representative set $\hat{P}^r_{s,v}$ in line 2 (for $r = A_1 \cdots A_k$) and subsequently iterates over each set of nodes X in $\hat{P}^r_{s,v}$ to test if there is a path from v to t that avoids X.

For the correctness of the algorithm, the next lemma is crucial.

LEMMA 4.12. *Let $r_1 T^*$ be a 0-bordered expression with $r_1 = A_1 \cdots A_{k_1}$ and let $L(r_2)$ be an arbitrary finite language with words up to length k_2. We define $k = k_1 + k_2$. Then, $G = (V, E)$ has a simple path from s to t that matches $r_1 T^* r_2$ if and only if there exists a node $v \in V$ and $X \in \hat{P}^{r_1}_{s,v} \subseteq^{k+1}_{rep} P^{r_1}_{s,v}$, such that G has a simple path from s to t that matches $r_1 T^* r_2$ and with the first $k_1 + 1$ nodes belonging to X.*

Proof. The if direction is straightforward. For the only-if direction, let $p = (v_0, a_1, v_1) \cdots (v_{n-1}, a_n, v_n)$ be a shortest simple path from s to t that matches $r_1T^*r_2$. We first give the intuition of the proof. We will partition p as depicted in Figure 3, depending on whether p is short or long. Here, p is the path consisting of the solid edges. Since P and Q are disjoint, we will find a path P' with $V(P') \in \hat{P}^{r_1}_{s,v}$ that is node-disjoint from Q. We then show that, if p is long, P' and R must be disjoint, otherwise it will contradict p being a shortest path.

More precisely, we make the following case distinction. If $|p| \le 2k_1 + k_2 + 1$, we define $P = (v_0, a_1, v_1) \cdots (v_{k_1-1}, a_{k_1}, v_{k_1})$ and $Q = (v_{k_1+1}, a_{k_1+2}, v_{k_1+2}) \cdots (v_{n-1}, a_n, v_n)$. Clearly, P matches r_1 and $(v_{k_1}, a_{k_1+1}, v_{k_1+1}) \cdot Q$ matches T^*r_2. We have that $V(P) \in P^{r_1}_{s,v_{k_1}}$, we have $|V(Q)| \le k+1$, and $V(P) \cap V(Q) = \emptyset$. Let $\hat{P}^{r_1}_{s,v_{k_1}}$ be a $(k+1)$-representative set of $P^{r_1}_{s,v_{k_1}}$. Then there exists a set $X \in \hat{P}^{r_1}_{s,v_{k_1}}$ with $X \cap V(Q) = \emptyset$. By definition of $P^{r_1}_{s,v_{k_1}}$, there exists a simple path P' from s to v_{k_1} with $V(P') = X$ that matches r_1. Therefore, $P' \cdot (v_{k_1}, a_{k_1+1}, v_{k_1+1}) \cdot Q$ is a simple path from s to t that matches $r_1T^*r_2$.

Otherwise, we have $|p| > 2k_1 + k_2 + 1$. We define $P = (v_0, a_1, v_1) \cdots (v_{k_1-1}, a_{k_1}, v_{k_1})$, $R = (v_{k_1+1}, a_{k_1+2}, v_{k_1+2}) \cdots (v_{n-k-2}, a_{n-k-1}, v_{n-k-1})$, and $Q = (v_{n-k}, a_{n-k}, v_{n-k+1}) \cdots (v_{n-1}, a_n, v_n)$. We thus have

$$p = P \cdot (v_{k_1}, a_{k_1+1}, v_{k_1+1}) \cdot R \cdot (v_{n-k-1}, a_{n-k}, v_{n-k}) \cdot Q.$$

Since p matches $r_1T^*r_2$, we furthermore know that P matches r_1, R matches T^*, and Q matches $T^*T^{k_1}r_2$.[12] Since $|V(Q)| = k+1$, $V(P) \in P^{r_1}_{s,v_{k_1}}$, and $V(P) \cap V(Q) = \emptyset$, the definition of $\hat{P}^{r_1}_{s,v_{k_1}} \subseteq^{k+1}_{\text{rep}} P^{r_1}_{s,v_{k_1}}$ guarantees, similar as in the previous case, the existence of a path P' from s to v_{k_1} that matches r_1 with $V(P') \in \hat{P}^{r_1}_{s,v_{k_1}}$ and $V(P') \cap V(Q) = \emptyset$. Let $P' = (v_0, a'_1, v'_1) \cdots (v'_{k_1-1}, a'_{k_1}, v_{k_1})$. If P' is disjoint from R, the path

$$p' = P' \cdot (v_{k_1}, a_{k_1+1}, v_{k_1+1}) \cdot R \cdot (v_{n-k-1}, a_{n-k}, v_{n-k}) \cdot Q$$

is a simple path matching $r_1T^*r_2$, and we are done.

We show that P' must be disjoint from R. Towards a contradiction, assume that there is an $i \in \{1, \ldots, k_1 - 1\}$[13] such that $v'_i = v_j \in V(R)$. We choose i minimal and build a new simple path $p' = (v_0, a'_1, v'_1) \cdots (v'_{i-1}, a'_i, v'_i)(v'_i, a_{j+1}, v_{j+2}) \cdots (v_{n-1}, a_n, v_n)$. This path matches $A_1 \cdots A_i T^* T^{k_1} r_2$. But since r_1T^* is 0-bordered, we have $T \subseteq A_i$ for all $1 \le i \le k_1$, so the new path matches $r_1T^*r_2$. Finally, we note that p' does not contain the edge $(v_{k_1}, a_{k_1+1}, v_{k_1+1})$, so p' is shorter than p, which contradicts the assumption that p was a shortest path from s to t that matches $r_1T^*r_2$. So P' must be disjoint from R. □

Notice that we allow $T = \emptyset$ in Lemma 4.12. Since $L(\emptyset^*) = \{\varepsilon\}$, this means that the lemma also deals with the case where the expression is just $A_1 \cdots A_{k_1}$. From the proof of Lemma 4.12 we can also infer the following corollary, which states that shortest matching paths can also be found with this method. It will be useful in Section 8.2.2 when considering enumeration problems.

Corollary 4.13. *Let r_1T^* be a 0-bordered expression with $r_1 = A_1 \cdots A_{k_1}$ and let $L(r_2)$ be an arbitrary finite language with words up to length k_2. We define $k = k_1 + k_2$. Then, $G = (V, E)$ has a simple path from s to t that matches $r_1T^*r_2$ if and only if there exists a node $v \in V$ and $X \in \hat{P}^{r_1}_{s,v} \subseteq^{k+1}_{rep} P^{r_1}_{s,v}$, such that G has a shortest simple path from s to t that matches $r_1T^*r_2$ and with the first $k_1 + 1$ nodes belonging to X.*

The following lemma states that Algorithm 1 is correct and runs in fixed parameter tractable time.

[12]The path Q does not necessarily match $T^{k_1}r_2$, since r_2 might contain words shorter than k_2.
[13]Since P and R are disjoint, we have v_0, $v_{k_1} \notin V(R)$.

ALGORITHM 2: Algorithm for 0-bordered STEs

Input: Graph $G = (V, E)$, nodes s, t in G, and 0-bordered regular expression $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$

Output: Does there exist a simple path from s to t matching r?

for *all* $v \in V$ **do**
  Compute $\hat{P}^{r_1}_{s,v} \subseteq^{k_1+k_2+1}_{\text{rep}} P^{r_1}_{s,v}$ in G with $r_1 = A_1 \cdots A_{k_1}$.
  for *all sets* $X \in \hat{P}^{r_1}_{s,v}$ **do**
    $V' \leftarrow (V \setminus X) \cup \{v\}$
    $E' \leftarrow E \cap (V' \times \Sigma \times V')$
    for *all* $u \in V'$ **do**
      Compute $\hat{P}^{r_2}_{u,t} \subseteq^{k_2+1}_{\text{rep}} P^{r_2}_{u,t}$ in (V', E') with $r_2 = A'_{k_2} \cdots A'_1$.
      for *all sets* $X' \in \hat{P}^{r_2}_{u,t}$ **do**
        $V'' \leftarrow (V' \setminus X') \cup \{u\}$
        $E'' \leftarrow E' \cap (V'' \times T \times V'')$ ▷ (V'', E'') has only T-edges
        if *there exists a path from* v *to* u *in* (V'', E'') **then**
          return YES

return NO

Lemma 4.14. *PSimPathExistence($\mathcal{R}$) is in FPT for the class $\mathcal{R}$ of 0-bordered STEs of the form $r = A_1 \cdots A_k T^*$. More precisely, it is in time $2^{O(|r|)} \cdot |E||V|^2$.*

Proof. The problem can be solved using Algorithm 1. Its correctness follows directly from Lemma 4.12 with $r_2 = \varepsilon$. Using Lemma 4.10, we now show that the algorithm is indeed an FPT algorithm.

We obtain from Lemma 4.10 that line 2 of Algorithm 1 takes $O\left(8^{k+o(k)}|E| \log |V| + |r||E|\right)$ time for each $v \in V$. Since we need to consider at most $|V| \cdot \binom{2(k+1)}{k+1} \cdot 2^{o(2(k+1))}$ sets X in line 3, the number of such sets we need to consider throughout the entire algorithm is at most $O(|V|4^{k+o(k)})$. Finally, line 6 can be checked by a reachability test (say, depth-first search) in time $O(|V| + |E|)$, so the overall running time is bounded by

$$O\left(|V| \cdot (8^{k+o(k)}|E| \log |V| + |r||E|) + 4^{k+o(k)} \cdot (|V|^2 + |E||V|)\right),$$

which is clearly in FPT for the parameter k. □

We now extend the algorithm to 0-bordered STEs of the form $A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$. Since STEs allow bounded expressions on both sides, we need to do more than simply apply Algorithm 1. Instead, we will use a nesting thereof, which we present in Algorithm 2. The next Lemma shows the correctness and running time of Algorithm 2.

Lemma 4.15. *Let $\mathcal{R}$ be the class of 0-bordered STEs of the form $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$. Then PSimPathExistence($\mathcal{R}$) is in FPT. More precisely, it is solvable in time $2^{O(|r|)} \cdot |V|^3|E|$.*

Proof. We prove that Algorithm 2 solves the problem in the required time. Recall that

$$P^r_{s,v} := \{V(p) \mid \text{ there is a simple path } p \text{ from } s \text{ to } v \text{ in } G \text{ that matches } r\}\ .$$

We first show correctness. Let $k = k_1 + k_2$. Obviously, $k \leq |r|$. Using Lemma 4.12 with $r_1 = A_1 \cdots A_{k_1}$ and $r_2 = A'_{k_2} \cdots A'_1$, it suffices to consider paths in which the first $k_1 + 1$ nodes belong to a set $X \in \hat{P}^{r_1}_{s,v} \subseteq^{k+1}_{\text{rep}} P^{r_1}_{s,v}$ for some $v \in V$. Then we need to find the rest of the path, that is, a simple path from v to t that matches $T^* A'_{k_2} \cdots A'_1$ and that does not use nodes in $X \setminus \{v\}$.

We can apply Lemma 4.12 on the graph obtained from (V', E') by reversing all edges and using the expression $A'_1 \cdots A'_{k_2} T^* \varepsilon$. Hence, if such a path exists in (V', E'), then there exists a node u such that its last $k_2 + 1$ nodes belong to a set $X' \in \hat{P}^{r_2}_{u,t} \subseteq^{k_2+1}_{\text{rep}} P^{r_2}_{u,t}$. It then remains to test if there is a path from v to u that matches T^* and avoids the nodes in $(X \cup X') \setminus \{u, v\}$, which is done in line 11. This concludes the correctness proof.

We next show that the algorithm is indeed in FPT. Lemma 4.10 allows us to compute, after the preprocessing phase which takes $O(|r||E|)$ time, $\hat{P}^{r_1}_{s,v}$ on line 2 in time $O(8^{k+o(k)}|E| \log |V|)$ and such that its size is at most $\binom{2k_1+k_2+2}{k_1+1} \cdot 2^{o(k_1+k_2)}$. Similarly, we can compute $\hat{P}^{r_2}_{u,t}$ on line 7 in time $O(8^{k+o(k)}|E| \log |V|)$ and such that its size is at most $\binom{2k_2+2}{k_2+1} \cdot 2^{o(k_2)}$.

This means that we need to consider $O(|V| \cdot 4^{k+o(k)})$ many sets in line 3. Computing $\hat{P}^{r_2}_{u,t}$ takes time $O\left(8^{k+1+o(k+1)}|E| \log |V|\right)$ for each $u \in V$, so we have $O(|V|^2 \cdot 4^{k+o(k)} \cdot 8^{k+o(k)}|E| \log |V|)$ time for this part and need to consider at most $O(|V|^2 \cdot 4^{k+o(k)} \cdot 4^{k+o(k)})$ many sets in line 8. Finally, the reachability test in line 11 is in $O(|V| + |E|)$, so in sum we obtain a running time of

$$O\Big(|r||E| + |V|^2 \cdot 4^{k+o(k)} \cdot \Big(8^{k+o(k)}|E| \log |V| + 4^{k+o(k)} \cdot (|V| + |E|)\Big)\Big).$$

□

The previous Lemma showed how to deal with 0-bordered STEs of the form $A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$. The next Lemma generalizes this to all 0-bordered STEs.

Lemma 4.16. *Let $\mathcal{R}$ be the class of 0-bordered STEs. Then PSimPathExistence($\mathcal{R}$) is in FPT. More precisely, it is solvable in time $2^{O(|r|)} \cdot |V|^3|E|$.*

Proof. We prove the lemma by case distinction on the form of r. Recall that

$$r = B_{\text{pre}} T^* B_{\text{suff}}.$$

We differentiate between the forms of B_{pre} and B_{suff}. There are two possible forms, that is (1) $B_1? \cdots B_\ell?$ with $\ell \geq 0$ or (2) $B_1 \cdots B_\ell$ with $\ell \geq 1$. If B_{pre} and B_{suff} are of form (1), the language of r is downward closed. Therefore the entire problem reduces to a reachability problem on a product between G and an NFA for r. According to Lemma 4.1, this problem can be solved in time $O(|G||r|)$, since it is possible to compute an NFA of size $|r|$ for each STE r.

If B_{pre} and B_{suff} are both of form (2), the result follows from Lemma 4.15, which internally uses Algorithm 2, in time $2^{O(|r|)} \cdot |V|^3|E|$. We now explain how Algorithm 2 can be changed to work if B_{pre} is of form (2) and B_{suff} of form (1). Assume we have $r = A_1 \cdots A_{k_1} T^* A'_{k_2}? \cdots A'_1?$. Then we replace everything from line 6 to line 12 with a test for a simple path from v to t matching the downward closed language $T^* A'_{k_2}? \cdots A'_1?$. The correctness is again by Lemma 4.12. For the running time we observe that testing if there is a simple path matching $T^* A'_{k_2}? \cdots A'_1?$ is in time $O(|G||r|)$ by Lemma 4.1, since the language is downward closed. The running time in this case is therefore

$$O\left(|r||E| + |V| \cdot \left(8^{|r|+o(|r|)}|E| \log |V| + 4^{|r|+o(|r|)} \cdot |G||r|\right)\right).$$

The case $r = A_1? \cdots A_{k_1}? T^* A'_{k_2} \cdots A'_1$ is symmetric. To see this, notice that it is equivalent to deciding if there is a simple path from t to s that matches the reverse of expression r in the graph G with all edges reversed. □

4.4 Main Upper Bound for Simple Paths

Lemma 4.17. *Let $c \in \mathbb{N}$ be a constant and let $\mathcal{R}$ be the class of STEs with cut border at most c. Then SimPathExistence($\mathcal{R}$) is in FPT. More precisely, it is in time $2^{O(|r|)} \cdot |V|^{c+3}|E|$.*

Proof. Let $r \in \mathcal{R}$ and let c_1 and c_2 be the left and right cut border of r, respectively. Hence, $r = A_1 \dots A_{c_1} r' A'_{c_2} \cdots A'_1$. (If $c_i = 0$, then the respective part of r is simply missing.) We can compute, for all $u, v \in V$, all paths p_1 from s to u matching $A_1 \cdots A_{c_1}$ and all paths p_2 from v to t matching $A'_{c_2} \cdots A'_1$ in time $O(|V|^c)$.[14] We then do a loop over all pairs (p_1, p_2) of such paths that are node-disjoint. For the remainder of the proof, fix such a pair (p_1, p_2). We delete in G all nodes in $(V(p_1) \setminus \{u\}) \cup (V(p_2) \setminus \{v\})$. In the remaining graph, we search a path from u to v that matches the rest of the regular expression. The rest r' can have one of the following forms.

- $r' = A_{c_1+1} \cdots A_{k_1} T^* A'_{k_2} \cdots A'_{c_2+1}$,
- $r' = A_1? \cdots A_{k_1}? T^* A'_{k_2} \cdots A'_{c_2+1}$,
- $r' = A_{c_1+1} \cdots A_{k_1} T^* A'_{k_2}? \cdots A'_1?$, or
- $r' = A_1? \cdots A_{k_1}? T^* A'_{k_2}? \cdots A'_1?$.

These are the only possibilities and each of them is 0-bordered. Thus, we can use Lemma 4.16, which allows us to solve PSimPathExistence(r') in time $2^{O(|r'|)} \cdot |V|^3|E|$. Since $|r'| \le |r|$, this shows that PSimPathExistence($\mathcal{R}$) is in FPT. So we need $2^{O(|r'|)} \cdot |V|^3|E|$ time for each set of nodes of size $c_1 + c_2$, and therefore have an overall time of $2^{O(|r|)} \cdot |V|^{c+3}|E|$. □

5 MAIN LOWER BOUND: PARAMETERIZED TWO DISJOINT PATHS

We prove our main lower bound by considering variants of the TwoDisjointPaths problem [26]. A *two-colored graph* is a directed graph in which every edge is labeled a or b. We consider the following parameterized problems:

- PTwoDisjointPaths: Given a graph G, nodes s_1, t_1, s_2, t_2, and parameter $k \in \mathbb{N}$, are there simple paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2 such that p_1 and p_2 are node-disjoint and p_1 has length k?
- PTwoColorDisjointPaths: Given a two-colored graph G, nodes s_a, t_a, s_b, t_b, and parameter $k \in \mathbb{N}$, is there a simple a-path p_a from s_a to t_a and a simple b-path p_b from s_b to t_b such that p_a and p_b are node-disjoint and p_a has length k?

It is well-known that TwoDisjointPaths, the non-parameterized version of PTwoDisjointPaths, is NP-complete [26].

We will prove that both PTwoColorDisjointPaths and PTwoDisjointPaths are W[1]-hard. The latter result is stronger, but we start by proving W[1]-hardness for PTwoColorDisjointPaths, because it makes the proof for PTwoDisjointPaths, which relies on it, easier to understand.

5.1 Two Colored Disjoint Paths

In this section, we prove the following theorem.

Theorem 5.1. *PTwoColorDisjointPaths is W[1]-hard.*

To prove the theorem, we use an adaptation of a proof of Slivkins [49, Theorem 2.1], who proved that k-Edge-Disjoint-Paths with parameter k is W[1]-hard in directed acyclic graphs (DAG). Furthermore, we use the idea of *control nodes* by Grohe and Grüber [28, Lemma 16], who showed that Slivkins' construction can be extended to show that k-Disjoint-Cycles is W[1]-hard.

Construction 1. (Construction of G_{col} and k_{col}.) Given an input instance (G, k) of k-clique, we construct a graph G_{col}, nodes s_a, t_a, s_b, t_b, and parameter k_{col} such that $(G, k) \in k$-clique if and only if $(G_{\text{col}}, s_a, t_a, s_b, t_b, k_{\text{col}}) \in$ PTwoColorDisjointPaths. Let n be the number of nodes of G. The

[14] For the purpose of the proof, it suffices to compute the paths without the edge labels here. For deciding whether there exists a simple path, it suffices to know that there exist node-disjoint simple paths matching $A_1 \cdots A_{c_1}$ and $A'_{c_2} \cdots A'_1$ and which nodes they use. We dropped the exact labels to have $O(|V|^c)$ complexity.

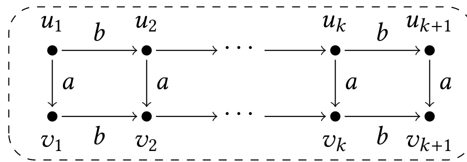

Fig. 4. Internal structure of each of the gadgets $G_{i,j}$.

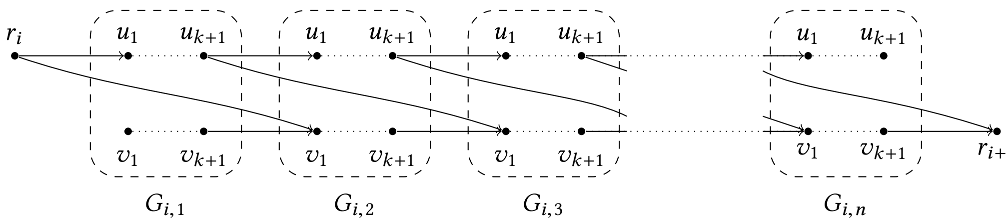

Fig. 5. The b-edges in row i. The internal structure of the $G_{i,j}$ is as in Figure 4.

graph G_{col} contains kn gadgets $G_{i,j}$ with $i = 1, \dots, k$ and $j = 1, \dots, n$, each consisting of $2(k+1)$ nodes. Gadgets will be ordered in k rows, where row i has the gadgets $G_{i,1}, \dots, G_{i,n}$. Furthermore, G_{col} contains $k+1$ additional nodes $r_1, \dots, r_{k+1}$ that link the rows together, and $k+1+k(k-1)/2$ *control nodes* $c_1, \dots c_{k+1}$ and c_{i_1,i_2} with $1 \le i_1 < i_2 \le k$ that will limit the number of disjoint paths from row $i-1$ to row i or from row i_1 to i_2, respectively. (The edge cases, c_1 and c_{k+1}, do not link rows together but just serve as start and end node, respectively.) We define $s_a = c_1$, $t_a = c_{k+1}$, $s_b = r_1$, and $t_b = r_{k+1}$. We will now explain how the nodes are connected in G_{col}. We will denote by $u \xrightarrow{a} v$ that there is an a-edge from u to v (similar for b-edges). Each gadget $G_{i,j}$ contains a disjoint copy of $2(k+1)$ nodes which we call $u_1, u_2, \dots, u_{k+1}$ and $v_1, v_2, \dots, v_{k+1}$. To simplify notation, we sometimes give these nodes the same name (e.g., in Figures 5, 6, and 7), even though they are different. One such gadget is depicted in Figure 4. To avoid ambiguity, we may also refer to node u_ℓ in gadget $G_{i,j}$ by $G_{i,j}[u_\ell]$. Each gadget contains edges $u_\ell \xrightarrow{a} v_\ell$ (for every $\ell = 1, \dots, k+1$) and $u_\ell \xrightarrow{b} u_{\ell+1}$ and $v_\ell \xrightarrow{b} v_{\ell+1}$ (for every $\ell = 1, \dots, k$).

We now explain how the gadgets $G_{i,j}$ are connected within the same row, see Figure 5. In each row $i \in \{1, \dots, k\}$, node r_i has two outgoing edges $r_i \xrightarrow{b} G_{i,1}[u_1]$ and $r_i \xrightarrow{b} G_{i,2}[v_1]$. We also have two incoming edges for r_{i+1}, namely $G_{i,n-1}[u_{k+1}] \xrightarrow{b} r_{i+1}$ and $G_{i,n}[v_{k+1}] \xrightarrow{b} r_{i+1}$. Furthermore, we have the edges $G_{i,j}[u_{k+1}] \xrightarrow{b} G_{i,j+1}[u_1]$ and $G_{i,j}[v_{k+1}] \xrightarrow{b} G_{i,j+1}[v_1]$ for every $j = 1, \dots, n-1$. We also add edges $G_{i,j}[u_{k+1}] \xrightarrow{b} G_{i,j+2}[v_1]$ for every $j = 1, \dots, n-2$. The latter edges ensure that every b-labeled path from r_i to r_{i+1} "skips" exactly one gadget $G_{i,j}$ for some $j = 1, \dots, n$.

We now explain how the gadgets $G_{i,j}$ are connected in different rows via the control nodes c_i and c_{i_1,i_2} (Figure 6). We first consider the edges from row i to $i+1$. In each row $i = 1, \dots, k-1$, and every $j = 1, \dots, n$, we add the edges $G_{i,j}[v_{k+1}] \xrightarrow{a} c_{i+1}$ and $c_{i+1} \xrightarrow{a} G_{i+1,j}[u_{i+2}]$. Furthermore, we add the edges $c_1 \xrightarrow{a} G_{1,j}[u_2]$ and $G_{k,j}[v_{k+1}] \xrightarrow{a} c_{k+1}$. We connect two rows i_1, i_2, with $1 \le i_1 < i_2 \le k$, by adding the edges $G_{i_1,j}[v_{i_2}] \xrightarrow{a} c_{i_1,i_2}$, and $c_{i_1,i_2} \xrightarrow{a} G_{i_2,j}[u_{i_1}]$ for all $j = 1, \dots, n$.

The edges of the original graph G are modeled in G_{col} by adding the edge $G_{i_2,x}[v_{i_1}] \xrightarrow{a} G_{i_1,y}[u_{i_2+1}]$ if and only if $1 \le i_1 < i_2 \le k$, $x \ne y$, and $(x, y) \in E$. This is illustrated in Figure 7.

Finally, we define $k_{\text{col}} = k(k-1)/2 \cdot 5 + 3k$. □

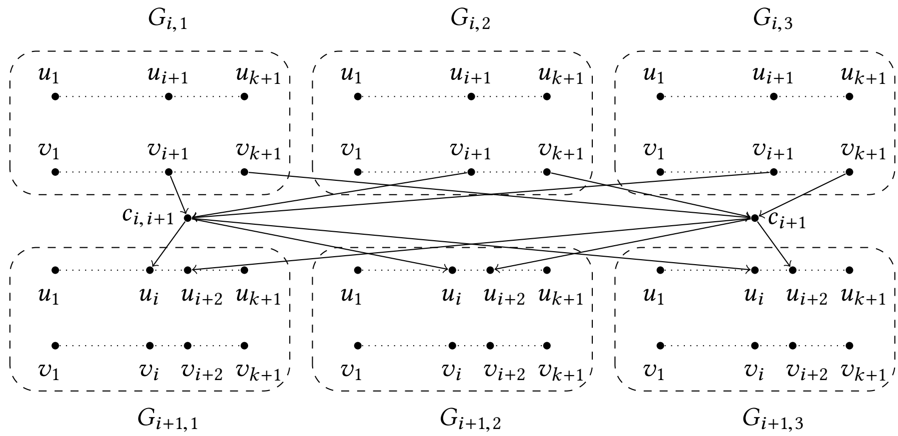

Fig. 6. The a-edges from row i to row $i+1$. (We assume $n = 3$ in the picture).

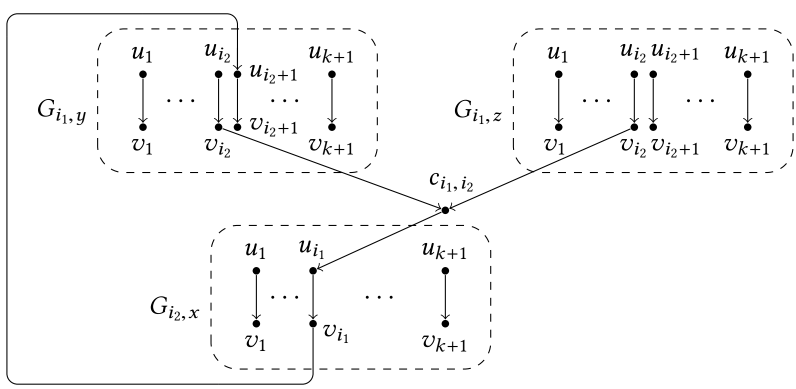

Fig. 7. The a-edges in the gadgets and between gadgets $G_{i_1,y}$, $G_{i_1,z}$ and $G_{i_2,x}$, with $i_1 < i_2 - 1$, under the assumption that $(x, y) \in E$ and $(x, z) \notin E$.

We denote by G^a_{col} the subgraph of G_{col} from Construction 1 that contains only the a-edges. We now prove a lemma that summarizes useful properties of G^a_{col}.

Lemma 5.2. *The graph G^a_{col} has the following properties:*

(a) G^a_{col} is a DAG. Moreover, there is a strict total order $<_c$ on all control nodes C such that, for every path from a node $v \in C$ to another node $v' \in C$ where no intermediate vertex is in C, node v' is the successor of v in $<_c$. The smallest and largest nodes in $<_c$ are c_1 to c_{k+1}, respectively.

(b) Each path in G^a_{col} from c_1 to c_{k+1} visits all control nodes, i.e., it contains all c_i and c_{i_1,i_2}, with $i \in \{1, \ldots, k+1\}$ and $1 \le i_1 < i_2 \le k$. Furthermore, it visits the control nodes in the order $<_c$.

(c) Each path in G^a_{col} has length at most k_{col}. Its length is exactly k_{col} if and only if it is from c_1 to c_{k+1}.

(d) Each path in G^a_{col} of length k_{col} has at least one edge $u_\ell \xrightarrow{a} v_\ell$ in every row of G^a_{col}.

Proof. First observe that G^a_{col} contains a fixed part that depends only on n and k, plus a set of edges that represent edges in G, i.e., edges that are present in G_{col} if and only if there exists a corresponding edge in G. Therefore, every possible graph G_{col} that the reduction produces is a subgraph of the case where G is a complete graph (i.e., if G has n nodes, it is the n-clique). Let G^a_{clique} denote the graph G^a_{col} in the case where G is the n-clique.

We first prove part (a). We show that, if G^a_{clique} has a cycle, then this cycle must contain a control node. Indeed, within the same row, the graph G^a_{clique} only has the edges from u_i to v_i in all the gadgets. So, there cannot be a cycle that only contains nodes from a single row. Therefore, the cycle must contain a path from some node in a row i_1 to a node in row i_2, for $i_1 < i_2$. Since every path in G^a_{clique} from row i_1 to i_2 with $i_1 < i_2$ contains at least one control node by construction, we have that every cycle in G^a_{clique} must contain a control node. It therefore remains to show that G^a_{clique} contains no cycle that uses a control node. To this end, observe that the relation $\prec$ where $n_1 \prec n_2$ if and only if $n_1 \neq n_2$ and n_2 is reachable from n_1 is a strict total order

$$c_1 \prec c_{1,2} \prec c_{1,3} \prec \ldots \prec c_{1,k} \prec c_2 \prec c_{2,3} \prec \ldots \prec c_{k-2,k} \prec c_{k-1,k} \prec c_k \prec c_{k+1} \qquad (\dagger)$$

on the control nodes C. That is, the order is such that control nodes are reachable in G^a_{clique} from all "smaller" control nodes and none of the "larger" control nodes. Notice that $\prec$ satisfies the requirements for $<_c$. Part (b) follows from (a). By (a), the smallest and largest nodes in $<_c$ are c_1 and c_{k+1}, respectively. Assume that p is a path from c_1 to c_{k+1}. Again by (a), p must visit every control node, in the order $<_c$.

We now prove part (c). First we prove that, between two consecutive[15] control nodes in G^a_{clique}, each path has a fixed length that depends only on the kind of control nodes. Then, since G^a_{clique} is a DAG by part (a), we can simply concatenate paths to obtain the length of paths from c_1 to c_{k+1}, showing (c). In this proof, when we consider a path that visits nodes in row i in G^a_{clique}, then by construction of G^a_{clique}, the length of this path is independent of the gadget $G_{i,j}$ that the path visits. That is, the path's length is the same for every $j = 1, \ldots, n$. To simplify notation, we therefore omit the j in $G_{i,j}[u]$ and write $G_i[u]$ instead.

We first consider the length of paths between consecutive control nodes in the ordering (†). Therefore, fix two such consecutive control nodes n_1 and n_2. We make a case distinction:

- $n_1 = c_i$ and $n_2 = c_{i,i+1}$: Each path from c_i to $c_{i,i+1}$ is of the form $c_i G_i[u_{i+1}] G_i[v_{i+1}] c_{i,i+1}$ and therefore has length 3.
- $n_1 = c_{i,j}$ and $n_2 = c_{i,j+1}$: Each path from $c_{i,j}$ to $c_{i,j+1}$ with $1 \leq i < j \leq k-1$ is of the form $c_{i,j} G_j[u_i] G_j[v_i] G_i[u_{j+1}] G_i[v_{j+1}] c_{i,j+1}$ and therefore has length 5.
- $n_1 = c_{i,k}$ and $n_2 = c_{i+1}$: Each path from $c_{i,k}$ to c_{i+1} is of the form $c_{i,k} G_k[u_i] G_k[v_i] G_i[u_{k+1}] G_i[v_{k+1}] c_{i+1}$ and therefore has length 5.
- $n_1 = c_k$ and $n_2 = c_{k+1}$: Each path from c_k to c_{k+1} is of the form $c_k G_k[u_{k+1}] G_k[v_{k+1}] c_{k+1}$ and therefore has length 3.

Since $\prec$ is a strict total order, this means that each path from c_1 to c_{k+1} in G^a_{clique} has the same length. We show that this length is exactly $k(k-1)/2 \cdot 5 + 3k = k_{\text{col}}$. The paths c_i to $c_{i,i+1}$ ($i = 1, \ldots, k-1$) and c_k to c_{k+1} sum up to length $3k$. For a fixed i we have $5 \cdot (k-i-1)$ paths from $c_{i,i+1}$ to $c_{i,k}$, which sum up to length $5(k(k-1)/2) - 5k + 5$ for $i = 1, \ldots, k-2$. Finally, we need to consider the paths from $c_{i,k}$ to c_{i+1}, which, for $i = 1, \ldots, k-1$, sum up to length $5k - 5$. This shows that each path G^a_{clique} from c_1 to c_{k+1} has length exactly k_{col}.

Since G^a_{clique} is a DAG and every node in G^a_{clique} is reachable from c_1, and c_{k+1} is reachable from all nodes and does not have outgoing edges in G^a_{clique}, the longest paths in G^a_{clique} are from c_1 to c_{k+1}. This shows (c).

[15] Control nodes x and y such that $x <_c y$ and there are no other control nodes in between.

Due to (b) and (c) each path of length k_{col} in G^a_{clique} contains c_i for $i = 1, \ldots, k+1$. Since each path from c_i to the next control node contains $(G_{i,j}[u_{i+1}], G_{i,j}[v_{i+1}])$, for a $j \in \{1, \ldots, n\}$ we also have (d). □

We are now ready to prove Theorem 5.1.

Proof of Theorem 5.1. We prove that $(G, k) \in k$-Clique if and only if $(G_{\text{col}}, s_a, t_a, s_b, t_b, k_{\text{col}}) \in$ PTwoColorDisjointPaths. Let us first assume that the undirected graph G has a k-clique with nodes $\{n_1, \ldots, n_k\}$. Then an a-path can go from c_1 to c_{k+1} using only the gadgets G_{i,n_i} with $i = 1, \ldots, k$. The reason is that, since $(n_{i_1}, n_{i_2}) \in E$, the edges $G_{i_2,n_{i_2}}[v_{i_1}] \xrightarrow{a} G_{i_1,n_{i_1}}[u_{i_2+1}]$ exist for all $i_1 < i_2$. Due to Lemma 5.2(c), this path has exactly k_{col} edges. The b-path, on the other hand, can go from r_1 to r_{k+1} and skip exactly G_{i,n_i} for all $i = 1, \ldots, k$ (using the diagonal edges in Figure 5). Since it skips these G_{i,n_i}, it is node-disjoint from the a-path and therefore we have a solution for PTwoColorDisjointPaths.

For the other direction let us assume that there exists a simple a-path p_a from c_1 to c_{k+1} and a simple b-path p_b from r_1 to r_{k+1} in G_{col} such that p_a and p_b are node-disjoint and p_a has length k_{col}. We show that G has a k-clique. Since every b-path from r_1 to r_{k+1} goes through each row, that is, from r_i to r_{i+1} for all $i = 1, \ldots, k$, this is also the case for p_b. By construction, p_b must also skip exactly one gadget in each row, using the diagonal edges in Figure 5. Indeed, this is the only way to move from r_i to r_{i+1} using only b-edges. Furthermore, for each gadget $G_{i,j}$ that p_b visits, it must be the case that it either visits all nodes $u_1, \ldots, u_{k+1}$ or all nodes $v_1, \ldots, v_{k+1}$. (This is immediate from Figure 4, showing all internal edges of a gadget.) Therefore, since p_a and p_b are node-disjoint, the path p_a cannot visit any gadget $G_{i,j}$ already visited by p_b. Therefore, p_a, which goes from c_1 to c_{k+1}, can only do so through the k skipped gadgets, call them G_{i,n_i} for $i = 1, \ldots, k$. Recall that the edges $G_{i_2,n_{i_2}}[v_{i_1}] \xrightarrow{a} G_{i_1,n_{i_1}}[u_{i_2+1}]$ with $i_1 < i_2$ only exist if $(n_{i_1}, n_{i_2}) \in E$. As these edges are necessary for the existence of the a-path from c_1 to c_{k+1} that uses only the skipped gadgets, all nodes n_i must be pairwise adjacent in G. That is, they form a clique of size k in G. □

5.2 Two Disjoint Paths

The two colors in the proof of Theorem 5.1 play a central role: since the a-path cannot use any b-edges and vice versa, we have much control over where the two paths can be. We now show that the construction in Theorem 5.1 can be strengthened so that we do not need the two colors. To this end, we replace the b-edges by long paths to ensure that all paths from s_a to t_a that have length at most k_{col} cannot use b-edges.

Construction 2. We construct the graph G_{node} from G_{col} by replacing each b-edge with a b-path of length k_{col}. (Even though PTwoDisjointPaths does not care about a-edges or b-edges, we keep them to simplify the reasoning in the remainder of the proof.) We define $s_1 = s_a$, $t_1 = t_a$, $s_2 = s_b$, and $t_2 = t_b$. (Notice that G_{col} has $O(k^2 n)$ nodes while G_{node} will have $O(k^2 n \cdot k_{\text{col}})$ nodes.) □

Lemma 5.3. *In G_{node}, we have that*

(a) every path from s_1 to t_1 has length at least k_{col} and
(b) every path from s_1 to t_1 has length exactly k_{col} if and only if it is an a-path.
(c) Furthermore, all properties of graph G^a_{col} from Lemma 5.2 also hold for G^a_{node}.

Proof. For part (a) we have two cases. If a path from s_1 to t_1 is an a-path, the result is immediate from Lemma 5.2(c). If it uses at least one b-edge, then it uses at least k_{col} b-edges by construction. Thus, the path will have length at least k_{col}.

For part (b), if a path from s_1 to t_1 has length exactly k_{col}, it uses at least one a-edge since t_1 only has incoming a-edges. If it used at least one b-edge, it would therefore use at least $k_{\text{col}} + 1$ edges which contradicts that the length is k_{col}. The converse direction is immediate from Lemma 5.2(c). The last point is obvious since G^a_{col} and G^a_{node} are the same. □

Lemma 5.4. *If $(G_{node}, s_1, t_1, s_2, t_2, k_{col}) \in$ PTwoDisjointPaths, then each solution p_1, p_2 is such that p_1 is an a-labeled path and p_2 a b-labeled path.*

Proof. It follows from Lemma 5.3 that p_1 can only use a-edges. We now show that the path p_2 from s_2 to t_2 can only use b-edges, that is, we show that it cannot use a-edges. There are three types of a-edges in G_{node}: (i) the ones from and to control nodes, (ii) "upward" edges that connect row i_2 to row i_1 with $i_1 < i_2$, and (iii) edges from u_ℓ to v_ℓ in one gadget.

Notice that, by construction, p_2 must visit nodes in row 1 and later also nodes in row k. To do so, p_2 cannot use edges from or to control nodes (type (i)), since, due to Lemma 5.2(b), p_1 already visits all control nodes. So p_2 cannot go from row i to a row j with $i < j$ via a-edges. This means that, if $i < j$, then p_2 can only go from row i to row j through r_{i+1} (and through nodes in row $i + 1$), since every remaining path from row i to a larger row goes through r_{i+1}. So, in order to go from row 1 to row k, path p_2 needs to visit all nodes $r_2, \ldots, r_k$, in that order. This means that it is also impossible for p_2 to use edges of type (ii). Indeed, if p_2 used an edge from row j to row i with $j > i$, then it would need to visit r_{i+1} a second time to arrive back in row j. Finally, if p_2 used an edge of type (iii) in row i, then, by construction, it would have to visit every gadget in this row. But since p_1 already uses at least one edge from u_ℓ to v_ℓ in each row, see Lemma 5.2(d), this means that p_2 cannot be node-disjoint with p_1. This completes the proof. □

Theorem 5.5. *PTwoDisjointPaths is W[1]-hard.*

Proof. We reduce from PTwoColorDisjointPaths, which is W[1]-hard due to Theorem 5.1.

We show that $(G_{\text{col}}, s_a, t_a, s_b, t_b, k_{\text{col}}) \in$ PTwoColorDisjointPaths if and only if $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}}) \in$ PTwoDisjointPaths. If $(G_{\text{col}}, s_a, t_a, s_b, t_b, k_{\text{col}}) \in$ PTwoColorDisjointPaths, then we can use the corresponding paths in G_{node} (where we follow the longer b-paths in G_{node} instead of the b-edges in G_{col}).

Conversely, if $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}}) \in$ PTwoDisjointPaths, it follows from Lemma 5.4 that the paths p_1 and p_2 correspond to paths p_a and p_b that are solutions for $(G_{\text{col}}, s_a, t_a, s_b, t_b, k_{\text{col}}) \in$ PTwoColorDisjointPaths. □

5.3 Main Lower Bound for Simple Paths

We are now ready to proof the hardness side of Theorem 3.5, i.e., Theorem 3.5(b).

Lemma 5.6. *Let $\mathcal{R}$ be a class of STEs that can be sampled. If $\mathcal{R}$ is not cuttable, then the problem PSimPathExistence($\mathcal{R}$) is W[1]-hard.*

Proof. Let $\mathcal{R}$ be an arbitrary but fixed class of STEs that is not cuttable and that can be sampled. We show that PSimPathExistence($\mathcal{R}$) is W[1]-hard by giving an FPT reduction from PTwoDisjoint-Paths restricted to instances of the form $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}})$ from Construction 2. The problem PTwoDisjointPaths is W[1]-hard due to Theorem 5.5.

Consider an input $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}})$ of PTwoDisjointPaths. We will construct an input $(G_{\text{lab}}, s, t, r)$ for PSimPathExistence($\mathcal{R}$) such that $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}}) \in$ PTwoDisjointPaths if and only if $(G_{\text{lab}}, s, t, r) \in$ PSimPathExistence($\mathcal{R}$).

Since $\mathcal{R}$ is not cuttable and can be sampled, a k'-bordered expression $r \in \mathcal{R}$ for some $k' \geq 2k_{\text{col}} + 1$ can be computed within time $f(k_{\text{col}})$, for some computable function f. Since r can be computed in time $f(k_{\text{col}})$, we know that $|r| \leq f(k_{\text{col}})$. Let k_{lab} be the maximum of the left and right cut border of r. Since k' is the sum of the left and right cut borders, $k_{\text{lab}} \geq k_{\text{col}} + 1$. Here we only consider the case that the left cut border is k_{lab}, i.e., $T \not\subseteq A_{k_{\text{lab}}}$, the other case is symmetric. We therefore know that r is of the form

$$r = A_1 \cdots A_{k_{\text{col}}} \cdots A_{k_{\text{lab}}} \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1 \quad \text{or} \quad r = A_1 \cdots A_{k_{\text{col}}} \cdots A_{k_{\text{lab}}} \cdots A_{k_1} T^* A'_{k_2}? \cdots A'_1? \,.$$

We now construct (G_{lab}, s, t). Fix three words w_1, w_2, and w_3 such that

- $w_1 \in L(A_1 \cdots A_{k_{\text{col}}})$,
- $w_2 \in L(A_{k_{\text{col}}+1} \cdots A_{k_{\text{lab}}} \cdots A_{k_1})$, and
- $w_3 \in L(A'_{k_2} \cdots A'_1)$.[16]

Notice that such words indeed exist. For the construction of G_{lab}, we start with the graph G_{node}. The main idea is to have at most one edge with a label in $A_{k_{\text{lab}}}$ that is reachable from s by a path of length $k_{\text{lab}} - 1$. More formally, fix an $x \in (T \setminus A_{k_{\text{lab}}})$, which must exist due to choice of k_{lab}.

- We replace each b-edge in G_{node} with an x-path of length k_{lab} (using $k_{\text{lab}} - 1$ new nodes for each replacement). We need to do this, because k_{lab} is potentially much larger than k_{col}.
- We change the labels of the a-edges in G_{node} such that each path from s_1 to t_1 is labeled w_1. Notice that the label for each such edge is well-defined. Indeed, by Lemma 5.2(c) we have that each a-path from s_1 to t_1 has length exactly k_{col}. If there were an edge e on an a-path from s_1 to t_1 that is reachable from s_1 through n_1 edges and also through n_2 edges, with $n_1 \neq n_2$, then, since t_1 is reachable from e, it means that there would be paths of different lengths from s_1 to t_1.
- We add a path labeled w_2 from t_1 to s_2. We refer to this path as *the* w_2*-labeled path* in the remainder of the proof.
- We add a path labeled w_3 from t_2 to a new node t, to which we will refer as the w_3*-labeled path* in the remainder of the proof.

The resulting tuple $(G_{\text{lab}}, s_1, t, r)$ serves as input for PSimPathExistence($\mathcal{R}$). This concludes the reduction.

We now show that the reduction is correct. Therefore, we show that $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}}) \in$ PTwoDisjointPaths if and only if $(G_{\text{lab}}, s_1, t, r) \in$ PSimPathExistence($\mathcal{R}$). If $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}})$ $\in$ PTwoDisjointPaths with solution p_1 and p_2, then there exists a (unique) simple path from s_1 to t in G_{lab} that contains the nodes $V(p_1) \cup V(p_2)$ and matches r.

Conversely, if $(G_{\text{lab}}, s_1, t, r) \in$ PSimPathExistence($\mathcal{R}$), then there exists a simple path p from s_1 to t in G_{lab} that matches r. We will now prove the following:

(i) Consider the graph G^a_{node}, obtained from G_{node} by deleting all b-edges and nodes that have no adjacent a-edges. The nodes of $p[0, k_{\text{col}}]$ form a simple path from s_1 to t_1 in G^a_{node}.
(ii) The path $p[0, k_1]$ ends in s_2 and is labeled $w_1 w_2$.
(iii) The path p is labeled $w_1 w_2 w' w_3$ with $w' \in L(T^*)$. Its suffix of length $|w_3|$ starts in t_2 and ends in t.
(iv) The subpath of p from s_2 to t_2 is an x-path.

We prove (i). By definition of r, the edge $p[k_{\text{lab}} - 1, k_{\text{lab}}]$ is labeled by some symbol in $A_{k_{\text{lab}}}$. Therefore, this symbol cannot be x. By construction of G_{lab}, this edge is either an edge that was labeled a in G_{node}, an edge on the w_2-labeled path, or an edge on the w_3-labeled path (since all other edges are labeled x).

The w_3-labeled path is not reachable from s_1 with a path of length smaller than k_{lab}, so this cannot be the case. Furthermore, the w_2-labeled path starts in t_1 and is therefore only reachable with a path of length at least k_{col} (see Lemma 5.3), so we can also exclude that. Therefore, the first $k_{\text{col}} + 1$ nodes must form an a-path in G_{node}. From Lemma 5.2(c), we know that each path in G^a_{node}

[16]We use $w_3 \in L(A'_{k_2} \cdots A'_1)$ in case that r ends with $A'_{k_2} \cdots A'_1$ but also if it ends with $A'_{k_2}? \cdots A'_1?$.

of length k_{col} goes from s_1 to t_1 which implies (i). Since all nodes (except s_2) that belong to the w_2-labeled path of length $k_1 - k_{\text{col}}$ have only one outgoing edge, we have that $p[0, k_1]$ ends in s_2 and must match $w_1 w_2$. This shows (ii).

Since p matches $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ or $r = A_1 \cdots A_{k_1} T^* A'_{k_2}? \cdots A'_1?$, and since each word in $A_1 \cdots A_{k_1}$ has length k_1, it follows that $\text{lab}(p) = w_1 w_2 w'$ with $w' \in L(T^* A'_{k_2} \cdots A'_1) \cup L(T^* A'_{k_2}? \cdots A'_1?)$.

By construction of G_{lab}, the w_3-labeled path is the unique path of length $|w_3|$ leading to t. Therefore, each path from s_1 to t in G_{lab} must end with the w_3-labeled path which is from t_2 to t. Since $w_3 \in L(A'_{k_2} \cdots A'_1)$ and $|w_3|$ is the length of every word in $L(A'_{k_2} \cdots A'_1)$, we have that $\text{lab}(p) = w_1 w_2 w' w_3$ where $w' \in L(T^*)$. So we have (iii). Let p' be the part of p labeled w'. It follows from (ii) and (iii) that p' is a path from s_2 to t_2. Since it must be node-disjoint from $p[0, k_{\text{col}}]$, which is entirely in G^a_{node}, it follows from Lemma 5.4 that p' cannot use edges that correspond to ones in G^a_{node}.

Therefore, p' consists only of edges labeled x. This shows that G_{node} and k_{col} are in PTwoDisjointPaths, because $p[0, k_{\text{col}}]$ corresponds to a path p_1 and p' to p_2, which are solutions to PTwoDisjointPaths.

Finally, we note that the construction can indeed be done in FPT since the expression $r \in \mathcal{R}$ can be determined in time $f(k_{\text{col}})$ for a computable function f, and all changes we made to the graph are in time $h(k_{\text{col}}) \cdot |G_{\text{node}}|$, for a computable function h, which is FPT. Indeed, we only relabeled all edges, replaced each edge at most once with k_{lab} new edges and added other paths of length at most $|r|$. Since $|r| \leq f(k_{\text{col}})$, we indeed have an FPT reduction. □

6 CONNECTION BETWEEN SIMPLE PATHS AND TRAILS

LaPaugh and Rivest [33, Lemma 1 and Lemma 2] and Perl and Shiloach [46, Theorem 2.1 and Theorem 2.2] showed that there is a strong correspondence between trail and simple path problems that we will use extensively and therefore revisit here. Since the statements of the results do not precisely capture what we need, we have to be a bit more precise.

The Split Graph. The following construction is from LaPaugh and Rivest [33, Proof of Lemma 1]. Let $(G, s_1, t_1, \ldots, s_k, t_k)$ be a graph G together with nodes $s_1, t_1, \ldots, s_k, t_k$. We define $\text{split}(G, s_1, t_1, \ldots, s_k, t_k)$ as the tuple $(G', s'_1, t'_1, \ldots, s'_k, t'_k)$ obtained as follows. The graph G' is obtained from G by replacing each node v by two nodes v^{in} and v^{out}. A directed edge is added from v^{in} to v^{out}. All incoming edges of v become incoming edges of v^{in} and all outgoing edges of v become outgoing edges of v^{out}. For every s_i and t_i, we define $s'_i = s^{\text{in}}_i$ and $t'_i = t^{\text{out}}_i$. For a path $p = (u_1, u_2) \cdots (u_{n-1}, u_n)$ in G, denote by $\text{split}(p)$ the path

$$(u^{\text{in}}_1, u^{\text{out}}_1)(u^{\text{out}}_1, u^{\text{in}}_2) \cdots (u^{\text{out}}_{n-1}, u^{\text{in}}_n)(u^{\text{in}}_n, u^{\text{out}}_n).$$

The following Lemma is immediate from LaPaugh and Rivest's construction.

Lemma 6.1. *Let $(G', s'_1, t'_1, \ldots, s'_k, t'_k) = \text{split}(G, s_1, t_1, \ldots, s_k, t_k)$. Then the following hold:*

(1) For every $i = 1, \ldots, n$, the path $p = (s_i, u_1) \cdots (u_n, t_i)$ is a simple path from s_i to t_i in G if and only if split(p) is a trail in G'.

(2) For every $i = 1, \ldots, n$, the number of simple paths from s_i to t_i in G equals the number of trails from s'_i to t'_i in G'.

(3) There exist pairwise node disjoint simple paths of length k_i from s_i to t_i in G for every $i = 1, \ldots, k$ if and only if there exist pairwise edge disjoint trails of length $2k_i + 1$ from s'_i to t'_i in G' for every $i = 1, \ldots, k$.

The Line Graph. We denote by line$(G, s_1, t_1, \ldots, s_k, t_k)$ a variation on the *line graph of* G [33, Proof of Lemma 2]. The line graph construction was used by LaPaugh and Rivest to reduce the edge disjoint subgraph homeomorphism problem to the node disjoint subgraph homeomorphism problem. More precisely, we denote by line$(G, s_1, t_1, \ldots, s_k, t_k)$ the tuple $(G', s_1, t_1, \ldots, s_k, t_k)$ obtained as follows. Let $G = (V, E)$. The nodes of G' are $\{v_{(u_1,u_2)} \mid (u_1, u_2) \in E\} \cup \{s_1, t_1, \ldots, s_k, t_k\}$. The edges of G' are the disjoint union of

- $\{(v_{(u_1,u_2)}, v_{(u_2,u_3)}) \mid (u_1, u_2)$ and $(u_2, u_3) \in E\}$,
- $\{(s_i, v_{(s_i,u)}) \mid i = 1, \ldots, k$ and $(s_i, u) \in E\}$, and
- $\{(v_{(u,t_i)}, t_i) \mid i = 1, \ldots, k$ and $(u, t_i) \in E\}$.

Lemma 6.2. *Let $(G', s_1, t_1, \ldots, s_k, t_k) = line(G, s_1, t_1, \ldots, s_k, t_k)$. Then the following hold:*

(1) For every $i = 1, \ldots, n$, the path $(s_i, u_1) \cdots (u_n, t_i)$ is a trail from s_i to t_i in G if and only if $(s_i, v_{(s_i,u_1)})(v_{(s_i,u_1)}, v_{(u_1,u_2)}) \cdots (v_{(u_{n-1},u_n)}, v_{(u_n,t_i)})(v_{(u_n,t_i)}, t_i)$ is a simple path in G'.

(2) For every $i = 1, \ldots, n$, the number of trails from s_i to t_i in G equals the number of simple paths from s_i to t_i in G'.

(3) There exist pairwise edge-disjoint trails of length k_i from s_i to t_i in G for every $i = 1, \ldots, k$ if and only if there exist pairwise node-disjoint simple paths of length $k_i + 1$ from s_i to t_i in G' for every $i = 1, \ldots, k$.

Proof. Properties (1) and (2) are immediate from the construction. Property (3) follows from (1): if we have edge-disjoint trails, then the same simple paths as obtained in (1) are node-disjoint and the other way around. If they were not node-disjoint, at least two would share a node, say, $v_{(u_1,u_2)}$ in G', but they only contain this node both if the corresponding trails in G have the edge (u_1, u_2), so the trails in G wouldn't be edge-disjoint. □

Adding Edge Labels. If we additionally consider edge labels and RPQs, the correspondence between simple paths and trails is a bit more complex. We prove here that upper bounds transfer from simple path problems to trail problems. This would be a version of Lemma 6.2 for labeled graphs.

Notice that strengthening Lemma 6.1 for labeled graphs without changing the language of the RPQ is impossible if FPT ≠ W[1]. To see this, we note that the expression $a^k b^*$ is conflict-free, but not cuttable. This implies that PTrailExistence$(a^k b^*)$ is in FPT while PSimPathExistence$(a^k b^*)$ is W[1]-hard (see Theorem 3.5 and Theorem 3.7). Since a strengthened version of Lemma 6.1 would imply that PSimPathExistence$(a^k b^*)$ is at most as hard as PTrailExistence$(a^k b^*)$, such a lemma can only exist when FPT = W[1].

Lemma 6.3. *Let r be an RPQ, let σ be an arbitrary symbol in Σ, let G be a graph with labels in Σ, and s, t nodes in G. Then there exist a graph H and nodes s', t' such that there exists a trail from s to t in G that matches r if and only if there exists a simple path from s' to t' in H that matches the RPQ $\sigma \cdot r$. Furthermore, H, s', and t' can be computed using logarithmic space and $H = (V_H, E_H)$ with $|V_H| = O(|E|)$ and $|E_H| = O(|E|^2)$.*

Proof. Given G, s, and t, we will construct a graph H and nodes s' and t' such that there exists a simple path from s' to t' in H matching the RPQ $\sigma \cdot r$ if and only if there exists a trail from s to t matching r in G. In fact, $(H, s', t') = line(G, s, t)$ with labels. More precisely, let $\sigma \in \Sigma$ be fixed. Let $H = (V_H, E_H)$ with $V_H = \{v_e \mid e \in E\} \cup \{s', t'\}$ and $E_H = \{(v_{(u_1,a_1,u_2)}, a_1, v_{(u_2,a_2,u_3)}) \mid (u_1, a_1, u_2), (u_2, a_2, u_3) \in E\} \cup \{(s', \sigma, v_{(s,a,u)}) \mid (s, a, u) \in E\} \cup \{(v_{(u,a,t)}, a, t') \mid (u, a, t) \in E\}$. An example of this reduction can be seen in Figure 8. From this construction, it immediately follows that $|V_H| = O(|E|)$ and $|E_H| = O(|E|^2)$.

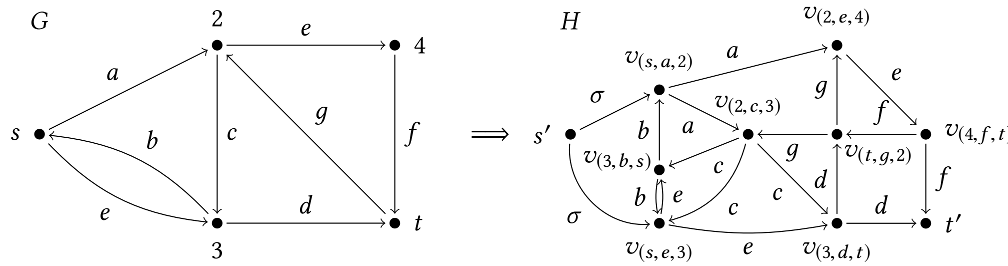

Fig. 8. Example of a part of the reduction in Lemma 6.3. There exists a trail from s to t matching r in the left graph G if and only if there exists a simple path from s' to t' matching $\sigma \cdot r$ in the right graph H. So we effectively have to find a simple path that matches r in the right graph (H) from $s'_1 = v_{(s,a,2)}$ to t' or from $s'_2 = v_{(s,e,3)}$ to t'.

We argue that this construction is correct. Indeed, assume there exists a path

$$p = (s, a_0, v_1)(v_1, a_1, v_2) \cdots (v_k, a_k, t)$$

from s to t in G that matches r and has pairwise disjoint edges. Then the path

$$p' = (s', \sigma, v_{(s,a_0,v_1)})(v_{(s,a_0,v_1)}, a_0, v_{(v_1,a_1,v_2)})(v_{(v_1,a_1,v_2)}, a_1, v_{(v_2,a_2,v_3)}) \cdots (v_{(v_k,a_k,t)}, a_k, t')$$

is a simple path from s' to t' in H that matches $\sigma \cdot r$. The other direction follows analogously since each path from s' to t' in H that matches $\sigma \cdot r$ has this form and we can therefore find the corresponding path from s to t in G. □

We note that, in the proof of Lemma 6.3, there is a clear correspondence between nodes in H and edges in G. To be more precise, each node in H, except for s' and t', corresponds to exactly one edge in G. We therefore obtain the following corollary:

Corollary 6.4. *Let r be an RPQ, G a graph, and s, t nodes in G. Let (H, s', t') and $\sigma \cdot r$ be the instance obtained from G, s, and t as in Lemma 6.3. Then there exists a bijection f_{sp} from the set of trails from s to t in G to the set of simple paths from s' to t' in H such that $\sigma \cdot lab(p) = lab(f_{sp}(p))$. Moreover, f_{sp} and f_{sp}^{-1} are computable in linear time.*

Proof. Let $p = (s, a_1, u_1)(u_1, a_2, u_2) \cdots (u_{n-1}, a_n, t)$ be a trail in G. Then we define $f_{sp}(p) = (s', \sigma, v_{(s,a_1,u_1)})(v_{(s,a_1,u_1)}, a_1, v_{(u_1,a_2,u_2)})(v_{(u_1,a_2,u_2)}, a_2, v_{(u_2,a_3,u_3)}) \cdots (v_{(u_{n-1},a_n,t)}, a_n, t')$ in H. Since all edges in p are pairwise different, the nodes $v_{(s,a_1,u_1)}, v_{(u_1,a_2,u_2)}, \ldots, v_{(u_{n-1},a_n,t)}$ (and s' and t') must be pairwise different. The mapping f_{sp} is a bijection since each simple path p' from s' to t' in H has such a form and we can therefore find the corresponding unique path $f_{sp}^{-1}(p')$ from s to t in G. □

7 EVALUATION FOR TRAILS

In this section, we prove Theorem 3.7. To this end, we first consider the following fundamental parameterized problems for trails:

- PTrailLength: Given a graph G, nodes s and t, and parameter $k \in \mathbb{N}$, is there a trail from s to t of length exactly k in G?
- PTrailLength$^{\geq}$: Given a graph G, nodes s and t, and parameter $k \in \mathbb{N}$, is there a trail from s to t of length at least k in G?

By Lemma 6.2, the complexities of Theorems 4.3 and 4.6 carry over from simple paths to trails.

Theorem 7.1. *PTrailLength and PTrailLength*$^{\geq}$ *are in FPT. More precisely, PTrailLength is in time* $2^{O(k)} \cdot |E|^3$ *and PTrailLength*$^{\geq}$ *in time* $2^{O(k)} \cdot |E|^4 \log |E|$.

Similarly, we can consider the trail version of the parameterized two disjoint paths problem, where we require the paths to be edge-disjoint trails.

- PTwoDisjointTrails: Given a graph G, nodes s_1, t_1, s_2, t_2, and parameter $k \in \mathbb{N}$, are there trails p_1 from s_1 to t_1 and p_2 from s_2 to t_2 such that p_1 and p_2 are edge-disjoint and p_1 has length k?

The following theorem is immediate from Theorem 5.5 and Lemma 6.1(3).

Theorem 7.2. *PTwoDisjointTrails is W[1]-hard.*

Next we will prove our main dichotomy for trails.

7.1 Upper Bound for Trails

Lemma 7.3. *Let $c \in \mathbb{N}$ be a constant and let $\mathcal{R}$ be the class of STEs with at most c conflict positions, that is, $\mathcal{R}$ is almost conflict-free. Then, PTrailExistence($\mathcal{R}$) is in FPT. More precisely, it is in time* $2^{O(|r|)} \cdot |E|^{c+6}$.

Proof. On graph G, we use the construction from the proof of Lemma 6.3 to obtain a graph $H = (V_H, E_H)$ such that there is a trail from s to t matching r in G if and only if there is a simple path from s' to t' matching $\sigma \cdot r$ in H (we can take σ to be an arbitrary label). So we need to decide whether there exists a simple path matching $\sigma \cdot r$ in H. To this end, we will do the following:

(1) We relabel the expression r to a conflict-free expression $\tilde{r}$. Then we enumerate all possible sets S of nodes of size up to c and relabel H depending on S, obtaining the graph H_S. We show that there is a simple path from s' to t' in H that matches $\sigma \cdot r$ if and only if there is a set S such that there is a simple path from s' to t' in H_S that matches $\sigma \cdot \tilde{r}$.
(2) Using a simple brute force algorithm, we can get rid of σ.
(3) We prove that Algorithm 2 does not only work for 0-bordered STEs, but also for conflict-free STEs when we restrict the graphs such that every node has only outgoing edges with the same label. Such graphs are obtained from the construction in Lemma 6.3. This allows us to use the methods from Lemma 4.16 to decide whether there exists a simple path matching $\tilde{r}$.

From (1)–(3) we can then conclude that deciding whether there exists a trail from s to t matching r with at most c conflict positions can be done using $|V_H|^{c+1}$ applications of Lemma 4.16, more precisely, $|V_H|^c$ times for all different sets S and $|V_H|$ times from the brute force algorithm to get rid of the σ. Since the time needed to find a simple path in Lemma 4.16 is $2^{O(|r|)} \cdot |V_H|^3 |E_H|$, and V_H and E_H are of size $O(|E|)$ and $O(|E|^2)$, respectively (Lemma 6.3), we obtain a running time of $2^{O(|r|)} \cdot |E|^{c+6}$.

We start with (1). Let $r_1 = B_{\text{pre}}$ and $r_2 = B_{\text{suff}}$ with $r = r_1 T^* r_2$. We change r_1 and r_2 by relabeling the labels in conflict positions. Let c_1 and c_2 denote the left and right cut borders of r. In r_1, we replace each conflict position A_i, where $i \leq c_1$, with $\tilde{A}_i$. Here, $\tilde{A}_i$ is $(A_i \setminus T) \cup \{\tilde{a} \mid a \in A_i \cap T\}$, where we assume w.l.o.g. that $\tilde{a}$ is a new symbol, not occurring in r. Analogously, we replace each A'_j, where $j \leq c_2$ with $\tilde{A}'_j$, where $\tilde{A}'_j = (A'_j \setminus T) \cup \{\tilde{a} \mid a \in A'_j \cap T\}$. We name the resulting expressions $\tilde{r}_1, \tilde{r}_2$, and $\tilde{r} = \tilde{r}_1 T^* \tilde{r}_2$ to avoid confusion. Notice that the relabeling affects only conflict positions, thus at most c many A_i or A'_j.

Then, we enumerate all subsets of up to c nodes in H. For each possible subset S, we generate the graph H_S by changing each edge (u, a, v) with $u \in S$ and $a \in T$ to $(u, \tilde{a}, v)$.

We prove that there is a simple path from s' to t' in H that matches $\sigma \cdot r$ if and only if there is a set S such that there is a simple path from s' to t' in H_S that matches $\sigma \cdot \tilde{r}$. Assume that there is a simple path $p = (s', \sigma, v_1)(v_1, a_1, v_2) \cdots (v_\ell, a_\ell, t')$ from s' to t' in H that matches $\sigma \cdot r$. We choose $I_1 = \{i \mid a_i \in A_i \cap T \text{ and } i \leq c_1\}$ and $I_2 = \{\ell + 1 - i \mid a_{\ell+1-i} \in A'_i \cap T \text{ and } i \leq c_2\}$ and $S = \{v_i \mid i \in I_1 \cup I_2\}$. Then, the path in H_S consisting of the same nodes as p, in the same order, is a simple path from s' to t' matching $\sigma \cdot \tilde{r}$ in H_S. Conversely, if there is a simple path from s' to t' matching $\sigma \cdot \tilde{r}$ in H_S, for some set S, the path using the same nodes in the same order in H will match $\sigma \cdot r$. This concludes (1).

For (2), we enumerate all nodes $v \in V_H$ with $(s', \sigma, v) \in E_H$. Since s' has no incoming edges by construction, we cannot reach s' (unless we start in s') and therefore we do not need to explicitly delete s'.

For (3), we prove in the online version of this article that Algorithm 2 also works for conflict-free STEs when the graphs are restricted to those where every node has only outgoing edges with the same label. Its proof is similar to the one of Lemma 4.12. The crucial part is that $\tilde{A}_1 \cdots \tilde{A}_{c_1}$ (where c_1 is the left cut border) and T have different labels. Since every node in H_S has only outgoing edges with the same labels, the first c_1 nodes of a path matching $\tilde{A}_1 \cdots \tilde{A}_{c_1}$ must therefore be node-disjoint from every path matching T^*.

Thus, we can use the methods[17] from Lemma 4.16 to decide whether, for any set S from (1) and node v from (2), there exists a simple path matching $\tilde{r}$ from v to t' in H_S. □

7.2 Lower Bound for Trails

Lemma 7.4. *Let $\mathcal{R}$ be a class of STEs that can be conflict-sampled. If $\mathcal{R}$ is not almost conflict free, then PTrailExistence($\mathcal{R}$) is W[1]-hard.*

Proof. The proof follows the lines of Lemma 5.6, i.e., we give a reduction from PTwoDisjointPaths. Let $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}})$ be an instance from PTwoDisjointPaths. Since $\mathcal{R}$ is not almost conflict-free and can be conflict-sampled, we can find an $r \in \mathcal{R}$ with at least $4k_{\text{col}} + 1$ conflict positions in time $f(k_{\text{col}})$, for some computable function f.

Let us assume that we have at least $2k_{\text{col}} + 1$ conflict positions in $A_1 \cdots A_{c_1}$, where c_1 is the left cut border of r. The case where we have at least $2k_{\text{col}} + 1$ conflict positions in $A'_{c_2} \cdots A'_1$ is symmetric. Therefore, r is of the form

$$A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1 \text{ or } A_1 \cdots A_{k_1} T^* A'_{k_2}? \cdots A'_1?$$

Starting from G_{node}, we will now split the nodes as in Lemma 6.1, and relabel the graph depending on r. More precisely, fix three words w_1, w_2, and w_3 such that

- $w_1 = a_1 \cdots a_{c_1} \in L(A_1 \cdots A_{c_1})$, such that $|w_1| \geq 2k_{\text{col}} + 1$ and $a_i \in A_i \cap T$ in at least $2k_{\text{col}}$ positions $i \in \{1, \ldots, c_1 - 1\}$,
- $w_2 \in L(A_{c_1+1} \cdots A_{k_1})$, and
- $w_3 \in L(A'_{k_2} \cdots A'_1)$.[18]

Notice that, since $A_1 \cdots A_{c_1}$ has at least $2k_{\text{col}} + 1$ conflict positions, we can indeed choose w_1 such that $|w_1| \geq 2k_{\text{col}} + 1$ and $a_i \in A_i \cap T$ in at least $2k_{\text{col}}$ positions with $i \leq c_1 - 1$. We will refer to the first $2k_{\text{col}}$ such positions as the *conflict indices of* w_1. If i is a conflict index, we refer to the symbol a_i as *conflict symbol.*

We explain how G_{node} is changed. By definition of cut borders, we have that $T \not\subseteq A_{c_1}$. So we can fix an $x \in (T \setminus A_{c_1})$.

- As in Lemma 6.1, we split each node v into v^{in} and v^{out}. Furthermore, if v has an adjacent (incoming or outgoing) a-edge in G_{node}, we label the edge from v^{in} to v^{out} with a. Otherwise,

[17]That is, depending on the form of $\tilde{r}$, we use a simple reachability test, Algorithm 2, or a mixture of both.
[18]We use $w_3 \in L(A'_{k_2} \cdots A'_1)$ in case that r ends with $A'_{k_2} \cdots A'_1$ but also if it ends with $A'_{k_2}? \cdots A'_1?$.

we label it b. Observe that the resulting graph is the split graph of G_{node}, with some additional labels. We therefore call the resulting graph split(G_{node}).
- We replace each b-edge of split(G_{node}) by an x-path of length c_1.
- We will now relabel the a-edges in split(G_{node}) such that the resulting paths from s_1^{in} to t_1^{out} match w_1. We do this in several steps. The conflict positions on w_1 play a crucial role in the graph and the substrings of w_1 between conflict indices will serve as "padding" on the paths. Recall that w_1 has exactly $2k_{\text{col}}$ conflict indices $\{i_1, \ldots, i_{2k_{\text{col}}}\}$. Furthermore, $2k_{\text{col}}$ is the length of every a-path from s_1^{in} to t_1^{in} in split(G_{node}) (due to the construction in Lemma 6.1 and Lemma 5.2(c)). Therefore, on each path from s_1^{in} to t_1^{in}, we can label the ℓ-th edge with the conflict symbol a_{i_ℓ} from w_1.
Since we only used the conflict indices of w_1 until now, we will still need to add padding to the paths to ensure that every path from s_1^{in} to t_1^{out} matches w_1. Furthermore, for the reduction to be correct, this padding needs to be done in a particular way, which we explain next. We label $(t_1^{\text{in}}, a_{c_1}, t_1^{\text{out}})$ with $a_{c_1} \in A_{c_1}$. (Since w_1 has $2k_{\text{col}} + 1$ conflict positions, c_1 is not a conflict index.) All paths from s_1^{in} to t_1^{out} are of the form
$$u_1^{\text{in}} u_1^{\text{out}} u_2^{\text{in}} u_2^{\text{out}} \cdots u_{k_{\text{col}}+1}^{\text{in}} u_{k_{\text{col}}+1}^{\text{out}}$$
for some nodes $u_1, \ldots, u_{k_{\text{col}}+1}$ from G_{node}. For the correctness of the reduction, it will be crucial that, for each $j = 2, \ldots, k_{\text{col}}$, the edge between u_j^{in} to u_j^{out} is labeled with a conflict symbol, so we can only replace the edges from u_j^{out} to u_{j+1}^{in} with longer paths. Therefore, for every $j = 1, \ldots, k_{\text{col}} - 1$, we replace each such edge $(u_j^{\text{out}}, a_{i_\ell}, u_{j+1}^{\text{in}})$ with a path labeled $w[i_{\ell-1} + 1, i_{\ell+1} - 1]$ (where all internal nodes on these paths are new). Notice that, for each such edge, we have that $2 \leq \ell \leq 2k_{\text{col}} - 1$, so $i_{\ell-1}$ and $i_{\ell+1}$ are indeed conflict indices of w_1. Additionally we replace $(u_{k_{\text{col}}}^{\text{out}}, a_{i_\ell}, u_{k_{\text{col}}+1}^{\text{in}})$ with the word $w_1[i_{2k_{\text{col}}}, |w| - 1]$. If the word $w_1[1, i_1 - 1]$ is non-empty, we replace the edge $(s_1^{\text{in}}, a_{i_1}, s^{\text{out}})$ with a new path labeled $w_1[1, i_1]$. As a result, every path from s_1^{in} to t_1^{out} is now labeled with w_1.
- We add a path labeled w_2 from t_1^{out} to s_2^{in}, which we will call *the w_2-labeled path*, and a path labeled w_3 from t_2^{out} to a new node t, which we will call *the w_3-labeled path*.

This completes the construction. Call the resulting graph G_{edge}.

We will now prove correctness, that is, $(G_{\text{node}}, s_1, t_1, s_2, t_2, k_{\text{col}})$ is a yes-instance from PTwo-DisjointPaths if and only if there is a trail from s_1^{in} to t matching r in G_{edge}.

For the direction from left to right, let $p_1 = u_1, \ldots, u_{k_{\text{col}}+1}$ be a simple path of length k_{col} from s_1 to t_1 and p_2 a simple path from s_2 to t_2 in G_{node}, such that p_1 and p_2 are node-disjoint. By Lemma 6.1 the path split(p_1) is a trail in split(G_{node}). By construction, there is a unique path P_1 from s_1^{in} to t_1^{out} in G_{edge} that contains all the edges of split(p_1). (Indeed, P_1 is the path split(p_1) with the extra padding.) Moreover, this path P_1 is a trail that matches w_1. Likewise, the path P_2 = split(p_2) is a trail in split(G_{node}) and, by construction, also a trail in G_{edge}. Moreover, it matches T^* because every edge is either labeled x or labeled with a conflict symbol. Since p_1 and p_2 are node-disjoint, P_1 and P_2 are also node-disjoint and therefore edge-disjoint. Finally, if P_{w_2} and P_{w_3} are the w_2- and w_3-labeled paths respectively, then $P_1 P_{w_2} P_2 P_{w_3}$ is a trail from s_1^{in} to t that matches r.

For the other direction, let p be a trail from s_1^{in} to t in G_{edge} that matches r. We need some additional notions. For a path p in G_{edge}, we denote by contract(p) the path in G_{node} obtained from p by removing the padding and contracting node pairs $(u^{\text{in}}, u^{\text{out}})$ back to u. Formally, if we view p as a sequence $u_1 \cdots u_n$ of nodes, such a path is obtained from p by removing all nodes except those in $\{u^{\text{out}} \mid u \in V_{\text{node}}\}$ and replacing each such node u^{out} by u. By definition of G_{edge}, the resulting sequence of nodes is indeed a path in G_{node}.

We will prove:

(i) The path $p_1 = \text{contract}(p[0, c_1])$ is a simple path from s_1 to t_1 in G_{node}. Moreover, p_1 has length k_{col} and each edge in p_1 is labeled a (so it is even a path in G^a_{node}).
(ii) The prefix of p of length k_1 ends in s_2^{in} and is labeled $w_1 w_2$.
(iii) The path p is labeled $w_1 w_2 w' w_3$ with $w' \in L(T^*)$. The w_3-labeled suffix of p starts in t_2^{out} and ends in t.

We prove (i). By definition of r, the edge $p[c_1 - 1, c_1]$ in G_{edge} is labeled by some symbol in A_{c_1}. Therefore, this symbol cannot be x. By construction of G_{edge}, the only edges that are not labeled x are either on some w_1-labeled path from s_1^{in} to t_1^{out}, on the w_2-labeled path, or on the w_3-labeled path. Since the w_3-labeled path is not reachable from s_1 by a path of length at most c_1 and the w_2-labeled path starts in t_1^{out} and is therefore only reachable from s_1 with a path of length at least c_1, the edge $p[c_1 - 1, c_1]$ must be on one of the w_1-labeled paths from s_1^{in} to t_1^{out}. Furthermore, the entire path $p[0, c_1]$ must be a prefix of some w_1-labeled path from s_1^{in} to t_1^{out}. Indeed, if this were not be the case, then $p[0, c_1]$ would have to contain an x-path of length c_1 (since we replaced every b-edge in $\text{split}(G_{\text{node}})$ by an x-path of length c_1), which is impossible because it is too short for that.

This means that $p_1 = \text{contract}(p[0, c_1])$ is indeed a path in G_{node} and every edge of p_1 is labeled a. Therefore, it is a path in G^a_{node}. Since $p[0, c_1]$ has precisely $2k_{\text{col}}$ conflict indices and additionally contains the edge $(t_1^{\text{in}}, a_{c_1}, t_1^{\text{out}})$, it contains precisely $2k_{\text{col}}+1$ edges of the form $(u^{\text{in}}, u^{\text{out}})$ or $(u^{\text{out}}, v^{\text{in}})$ for some nodes $u, v \in V_{\text{node}}$. Since, for each path p in G_{node}, the length of $\text{split}(p)$ is $2|p| + 1$, this means that the length of p_1 is precisely k_{col}. This implies (i).

Since all nodes that belong to the w_2-labeled path have only one outgoing edge, and since the path has length $k_1 - c_1$, we have that $p[0, k_1]$ ends in s_2^{in} and must match $w_1 w_2$. This shows (ii).

Since p matches $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ (the case $r = A_1 \cdots A_{k_1} T^* A'_{k_2}? \cdots A'_1?$ is analogous) and each word in $A_1 \cdots A_{k_1}$ has length k_1, it follows that $\text{lab}(p) = w_1 w_2 w'$ with $w' \in L(T^* A'_{k_2} \cdots A'_1)$. By construction of G_{edge}, the w_3-labeled path is the unique path of length $|w_3|$ leading to t. Therefore, each path from s_1^{in} to t in G_{edge} must end with the w_3-labeled path which is from t_2^{out} to t. Since $w_3 \in L(A'_{k_2} \cdots A'_1)$ and $|w_3|$ is the length of every word in $L(A'_{k_2} \cdots A'_1)$, we have that $\text{lab}(p) = w_1 w_2 w' w_3$ where $w \in L(T^*)$. So we have (iii).

Let p' be the part of p labeled w'. It follows from (ii) and (iii) that p' is a path from s_2^{in} to t_2^{out}. Let $p_2 = \text{contract}(p')$. First note that, by definition of G_{edge}, the resulting sequence of nodes is indeed a path in G_{node}. We show that p_1 and p_2 are node-disjoint. We first note that $p[0, c_1]$ and p' contain v^{in} if only if they contain v^{out}, since they start in s_1^{in} and s_2^{in} and end in t_1^{out} and t_2^{out}, respectively. Indeed, this is since v^{in} has only one outgoing edge and v^{out} only one incoming edge. So, if v^{out} belongs to $p[0, c_1]$, it cannot be part of p', otherwise $p[0, c_1]$ and p' both contain the edge $(v^{\text{in}}, v^{\text{out}})$, which would contradict that p is a trail. The same holds for nodes v^{out} that belong to p'. This implies that p_1 and p_2 cannot share a node and are therefore node-disjoint. Together with (i), we know that $|p_1| = k_{\text{col}}$, which implies that p_1 and p_2 are solutions to PTwoDisjointPaths.

Finally, we note that the construction can indeed be done in FPT since the expression $r \in \mathcal{R}$ can be determined in time $f(k_{\text{col}})$ for a computable function f, and all changes we made to the graph G_{node} are in time $h(k_{\text{col}}) \cdot |G_{\text{node}}|$, for a computable function h, which is FPT. Indeed, we only relabeled all edges, replaced each edge at most once with c_1 new edges, split each node at most once into two new ones, and added other paths of length at most $|r|$. Since $|r| \leq f(k_{\text{col}})$, we have an FPT reduction. □

8 ENUMERATION PROBLEMS

We now turn our attention to enumeration problems.

8.1 Enumeration of Arbitrary Paths and Shortest Paths

We first show that enumeration for arbitrary and shortest paths can be done in polynomial delay.

It is well known that PathExistence($\mathcal{R}$) is in PTIME for the complete class $\mathcal{R}$ of RPQs. Indeed, one only needs to construct the product of the graph (G, s, t) and an NFA N for the RPQ and test if (t, q_f) is reachable from (s, q_0), where q_0 and q_f are an initial and an accepting state of N, respectively. This favorable complexity carries over to EnumPaths and EnumShortPaths. At the core lies the following result by Ackerman and Shallit.

Theorem 8.1 (Theorem 3 in [1]). *Given an NFA N and a number $\ell \in \mathbb{N}$ in unary, enumerating the words in $L(N)$ of length ℓ can be done in polynomial delay.*

This result generalizes a result of Mäkinen [37], who proved that the words of length ℓ in $L(N)$ can be enumerated in polynomial delay if N is deterministic. Ackerman and Shallit genereralized this result to nondeterministic N and proved that, for a given length ℓ (which they call *cross-section*), the lexicographically smallest word in $L(N)$ can be found in time $O(|Q|^2\ell^2)$, where Q is the set of states on N. ([1], Theorem 1). They then prove that the set of all words of length ℓ can be computed in time $O(|Q|^2\ell^2 + |\Sigma||Q|^2x)$, where x is the sum of the length of the words of length ℓ ([1], Theorem 2). A closer inspection of their algorithm even shows that it has delay $O(|\Sigma||Q|^2|w|)$ where $|w|$ is the size of the next output. In fact, Ackerman and Shallit prove that the words in $L(N)$ can be enumerated in radix order.

It is easy to extend the algorithm of Ackerman and Shallit to solve EnumPaths in polynomial delay as follows. Assume that we want to enumerate the paths from s to t in G that match the RPQ r. We construct an NFA N_r for r and take the product with G. We interpret $G \times N_r$ as an NFA and define its set of initial states as $\{(s, i) \mid i \text{ is an initial state of } N_r\}$ and its set of accepting states as $\{(t, f) \mid f \text{ is an accepting state of } N_r\}$. The product automaton therefore has states (u, q) where u is a node from G and q a state from N_r. In the resulting automaton, we replace every transition $[(u_1, q_1), a, (u_2, q_2)]$ with $[(u_1, q_1), (u_1, a, u_2), (u_2, q_2)]$. Enumerating the words from the resulting automaton corresponds to enumerating the paths from s to t that match r. Using Theorem 8.1, we have the following corollary.

Corollary 8.2. *EnumPaths and EnumShortPaths can be solved in polynomial delay.*

For completeness, we note that counting the number of paths from s to t that match a given regular expression r is #P-complete in general, even if G is acyclic, see [36, Theorem 4.8(1)] and [5, Theorem 6.1].[19] The same holds for counting the number of shortest paths, since all paths in the proof of [36, Theorem 4.8(1)] have equal length.

8.2 Enumeration of Simple Regular Paths

We now turn to enumerating simple paths with polynomial delay. A starting point is Yen's algorithm [57] for enumerating simple paths from a source s to target t, without label constraints. Yen's algorithm usually takes another parameter K and returns the K shortest simple paths. In the online version of this article, we present a version for enumerating all simple paths and Algorithm 3 is a variant thereof that enumerates all simple paths that match a regular expression r, in order of increasing length.

We give a high-level explanation. We need the notion of *derivatives* of a language, see [16]. Given a language L and a word w, the *derivative of L w.r.t. w* is defined as

$$w^{-1}L := \{v \mid wv \in L\}.$$

[19] Arenas et al. [5] actually prove that the problem is spanL-complete. Although it is not known if spanL = #P, they are equal under Cook reductions.

ALGORITHM 3: Yen's algorithm with regular expressions

Input: Graph $G = (V, E)$, nodes s, t, regular expression r
Output: The simple paths from s to t in G that match r

1 $A \leftarrow \emptyset$ ▷ A is the set of paths already written to output
2 $B \leftarrow \emptyset$ ▷ B is a set of candidate paths from s to t
3 $p \leftarrow$ a shortest simple path from s to t in G that matches r
4 **while** $p \neq \mathit{null}$ **do**
5   **output** p
6   Add p to A
7   **for** $i = 0$ *to* $|p| - 1$ **do**
8         $G' \leftarrow (V', E')$, where $V' = V \setminus V(p[0, i-1])$ and $E' = E \cap (V' \times V')$ ▷ $V(p[0, -1]) = \emptyset$
9   **for** *every path* p_1 *in* A *with* $p_1[0, i] = p[0, i]$ **do**
10   Delete the edge $p_1[i, i+1]$ in G' ▷ This also deletes $p[i, i+1]$ since $p \in A$
        ▷ G' now no longer has paths already in A
11         $p_2 \leftarrow$ a shortest simple path from $p[i, i]$ to t in G' that matches $(\mathrm{lab}(p[0, i]))^{-1}L(r)$
12   Add $p[0, i] \cdot p_2$ to B
13     $p \leftarrow$ a shortest path in B ▷ $p \leftarrow$ null if $B = \emptyset$
14   Remove p from B

The first solution in Algorithm 3 is obtained by finding a shortest simple path p that matches r (line 3). The next simple path must differ in some edge from p. So we search (if it exists), for all i, a shortest simple path that shares the first i edges with p, but not the $(i + 1)$th edge. The first part of the path is identical to p, while the rest must match the derivative of L w.r.t. $\mathrm{lab}(p[0, i])$, i.e., it must match $(\mathrm{lab}(p[0, i]))^{-1}L(r)$. All paths found in this way match r and one of the shortest simple paths found this way is the next solution, which we again store in p. The next simple path must again differ in some edge from the paths we already found. So we search again, for all i, for a shortest simple path that shares the first i edges with the new p, but not the $(i + 1)$th edge. To avoid rediscovering an old path, we also forbid other edges to appear in the new path (lines 9–10).

In the next theorem, we state that Yen [57] showed that Algorithm 3 works without regular expressions, that is, for $r = \Sigma^*$.

Theorem 8.3 (Implicit in [57]). *Given a graph G, nodes s and t, and $r = \Sigma^*$, Algorithm 3 enumerates all simple paths from s to t in polynomial delay.*

Proof sketch. The original algorithm of Yen [57] finds, for a given G, s, t, and $K \in \mathbb{N}$, the K shortest simple paths from s to t in G. It has two differences with Algorithm 3, namely that it does not take regular expressions into account (or assumes that $r = \Sigma^*$) and that it stops when K paths are returned.

Let $G = (V, E)$. Yen does not prove that his algorithm has polynomial delay, but instead shows that the delay is $O(K|V| + |V|^3)$.[20] On lines 3 and 11, he uses an algorithm from [55] to find a shortest, and therefore simple, path in time $O(|V|^2)$. (Instead, one can also use Dijkstra's algorithm or breadth-first search.) Notice that the derivative $(\mathrm{lab}(p[0, i]))^{-1}L(r)$ on line 11 is again Σ^* since $r = \Sigma^*$.

Unfortunately, K can be exponential in $|G|$ in general. However, the reason why the algorithm has K in the complexity is line 9, which iterates over all paths in A. If we do not store A as a linked

[20] In [57], Section 5, he notes that computing path number k in the output costs, in his terminology, $O(KN)$ time in Step I(a) and $O(N^3)$ in Step I(b), with $N = |V|$.

list as in [57] but as a prefix tree of paths instead, the algorithm only needs $O(|V| + |E|)$ steps to complete the entire for-loop on line 9 (without any optimizations). Indeed, if paths p and p'' share the first i edges, they will share a path of length i from the root node in the prefix tree. So we can find all forbidden $(i + 1)$th edges by forbidding all edges that start at the end of this path. We therefore obtain delay $O(|V|^3 + |V||E|)$ from Yen's analysis. □

By inspecting Yen's correctness proof, one can infer the following.

Remark 1. *Yen's algorithm is also correct if we store an* arbitrary *simple path from s to t in p on line 3 and from $p[i, i]$ to t in G' in p_2 on line 11. For completeness, we provide a proof in the online version of this article.*

8.2.1 Enumeration for Downward Closed Languages. Yen's algorithm can easily be adapted to solve EnumSimPaths for regular languages, see Algorithm 3. In the case of languages that are closed under taking subsequences (downward closed languages), we will see that the algorithm even runs in polynomial delay. We want to make this more precise and also present a general method for using (variations of) Algorithm 3 for enumeration problems with time guarantees.

Remark 2. *Algorithm 3 makes two important calls to a black box algorithm for computing a shortest simple path that matches a regular language, namely on lines 3 and 11. (There is another mention of "shortest path" on line 13, but here we only need to find a shortest path stored in B. It is only important for the ordering of the outputs and not for the correctness of the algorithm.)*

We can generalize and formalize this remark as follows.

Lemma 8.4. *Let $\mathcal{R}$ be a class of regular expressions. If there exist algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ that, when given as input a graph G, nodes s and t, word w, and $r \in \mathcal{R}$, return in time $f(n)$ (with $f(n) \geq n$),*

(1) a simple path from s to t in G that matches $L(r)$ if it exists and "no" otherwise and

(2) a simple path from s to t in G that matches $w^{-1}L(r)$ if it exists and "no" otherwise

respectively, then EnumSimPaths($\mathcal{R}$) is in delay $O(|V|f(n))$ with preprocessing time $O(f(n))$, where $n = |G| + |r|$.

Furthermore, if $\mathcal{A}_1$ and $\mathcal{A}_2$ always return a shortest simple path (resp., a smallest simple path in radix order), then the enumeration can be done in order of increasing length (resp., in radix order), with the same time guarantees.

Proof. The algorithm for EnumSimPaths($\mathcal{R}$) consists of Algorithm 3, where we call $\mathcal{A}_1$ on line 3, algorithm $\mathcal{A}_2$ on line 11, and choose an arbitrary path, shortest path, or smallest path in radix order in B on line 13, depending on whether we want to enumerate in arbitrary order, order of increasing length, or radix order, respectively. The correctness follows from Remark 1.

Clearly, we need time $O(f(n))$ to output the first path (if it exists). Then, Algorithm 3 does up to $|V|$ iterations in line 7. If we use a prefix tree as a data structure for A, we can insert or find a path p in A in $O(|V|)$ time. Thus we can also find the right node in the prefix tree and then delete the up to $|E|$ many outgoing edge in G line 10 in $O(|V| + |E|)$. In line 11, we call algorithm $\mathcal{A}_2$.

In line 13 we need to find a minimal path among the candidates in B. If we again use a prefix tree as a data structure and start with $|p|$ instead of the first node in p, we can always output the leftmost path which is a minimal simple path. Finding and deleting are in time $O(|V|)$. Thus, we have a delay of $O(f(n))$ until the first output, and afterwards time $O(|V|(|V| + |E| + f(n)))$. □

We will now use Lemma 8.4 to infer upper bounds on EnumSimPaths.

Proposition 8.5. *Let $\mathcal{R}$ be the class of regular expressions defining downward closed languages. Then EnumSimPaths($\mathcal{R}$) is in polynomial delay, even when the paths need to be enumerated in radix order.*

Proof. We prove the existence of polynomial time algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ for the two problems in Lemma 8.4, from which the result follows. Proposition 4.2 guarantees that we can find a smallest path in radix order that matches r in time $O(|G|^2|r|^2|V|^2)$, which is sufficient for $\mathcal{A}_1$. For $\mathcal{A}_2$, it is easy to construct an NFA N with $L(N) = L(r)$ in polynomial time. We observe that, for each word w, the derivative $w^{-1}L(N)$ is again downward closed and we can compute an NFA for it in linear time (by simply redefining the set of initial states). After that, we can again use the algorithm from Proposition 4.2 to compute a smallest path in radix order. This concludes the description of $\mathcal{A}_2$. □

Using Lemma 6.3, we can immediately show that the upper bound from Lemma 8.5 also holds for trails.

Corollary 8.6. *EnumTrails is in polynomial delay for regular expressions r such that $L(r)$ is downward closed, even when the paths need to be enumerated in radix order.*

Proof. Given $r \in \mathcal{R}$ and a graph G. We use Lemma 6.3 to construct $(H, s_1, t_1), \dots, (H, s_n, t_n)$. The algorithm in Lemma 8.5 allows us to enumerate all simple paths from s_i to t_i in H that match r in radix order. Therefore, we use n parallel instances of this algorithm to enumerate, for all i, all simple paths from s_i to t_i in H in radix order. Since each simple path in each H corresponds to a trail in G, see Corollary 6.4, we can also output the corresponding trails in polynomial delay with radix order. □

8.2.2 Enumeration for Simple Transitive Expressions. We show that Theorem 3.5(a) — the FPT part — can be extended to enumeration problems. We do not need to prove any hardness results, since hardness for enumeration problems immediately follows from the hardness of their decision version, i.e., Theorem 3.5(b). To this end, a *parameterized enumeration problem* is defined analogously to an enumeration problem, but its input is of the form $(x, k) \in \Sigma^* \times \mathbb{N}$. It is in *FPT delay* if the preprocessing time (time before writing the first answer) and the time between writing every two consecutive answers is bounded by $f(k) \cdot |(x, k)|^c$ for a constant c and a computable function f. Notice that each problem in polynomial delay is also in FPT delay.

The goal of this section is to prove the following theorem.

Theorem 8.7. *Let $\mathcal{R}$ be a cuttable class of STEs. Then PEnumSimPaths($\mathcal{R}$) is in FPT delay, even when the paths need to be enumerated in radix order.*

This theorem immediately implies that the enumeration versions of PSimPathLength and PSimPathLength$^{\geq}$ (from Section 4.2) are in FPT delay.

Theorem 8.8. *PEnumSimPathLength and PEnumSimPathLength$^{\geq}$ are in FPT delay, even when the paths need to be enumerated in order of increasing length.*

We now turn to proving Theorem 8.7. In fact, the proofs of the enumeration results are all along the same lines and use Lemma 8.4. The FPT algorithms for the decision versions of the problems can be used as $\mathcal{A}_1$ in Lemma 8.4. We also show that we can provide $\mathcal{A}_2$. To this end, we will prove that each derivative language of an STE with cut border c is a union of STEs with cut border at most c (see Lemma 8.9). Finally, we prove that both algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ can be adjusted to return the smallest matching path in radix order if it exists.

We first show that derivatives of STEs are unions of STEs with at most the same cut border.

Lemma 8.9. *Let $w \in \Sigma^*$ and r be a c-bordered STE of size n. Then $w^{-1}L(r)$ is a union of STEs $r_1, \dots, r_m$ that can be computed in time $O(|w||r|)$ such that*

- *$m \leq n$ and*
- *each r_i is c'-bordered for some $c' \leq c$.*

Furthermore, if r is an STE with at most c conflict positions then, every STE in $w^{-1}L(r)$ also has at most c conflict positions.

Proof. Let $r = B_1 \cdots B_n$ be a c-bordered STE such that each B_i is either of the form A, $A?$, or T^* as in Definition 3.2. Let $w \in \Sigma^*$, $J_w = \{j \mid w \in L(B_1 \cdots B_j)\}$. Then a regular expression for $w^{-1}L(r)$ consists of the union

$$\Sigma_{j \in J_w} B_{j+1} \cdots B_n$$

and, if $w \in L(B_1 \cdots B_j)$ with $B_j = T^*$, we add $B_j \cdots B_n$ to the union.

Clearly, the union is of size at most n, and since each expression $B_{j+1} \cdots B_n$ or $B_j \cdots B_n$ is c'-bordered for some $c' \leq c$ by definition, the result follows. Since we can test $w \in L(r)$ in $O(|w||r|)$, we can compute J_w and therefore also the derivatives in $O(|w||r|)$. □

Example 8.10. For the regular expression $r = a^*aab$ and the word $w = aaa$, the derivative $w^{-1}L(r)$ is $\{a^*aab + aab + ab + b\}$.

Lemma 8.9 implies that we can strengthen Lemma 8.4 in the case of STEs.

Lemma 8.11. *Let $\mathcal{R}$ be a class of STEs. If there exists an algorithm $\mathcal{A}$ that, when given as input a graph G, nodes s and t, and $r \in \mathcal{R}$, returns in time $f(n)$ (with $f(n) \geq n$),*

a simple path from s to t in G that matches $L(r)$ if it exists and "no" otherwise,

then EnumSimPaths($\mathcal{R}$) is in delay $O(|V||r|f(n) + |r| \cdot |V|^3)$ with preprocessing time $O(f(n))$, where $n = |G| + |r|$. Furthermore, if $\mathcal{A}$ always returns a shortest simple path (resp., a smallest simple path in radix order), then the enumeration can be done in order of increasing length (resp., in radix order), with the same time guarantees.

Proof. Since we search for simple paths, we only need to compute derivatives for words w of length at most $|V|$. Lemma 8.9 implies that we can compute a single such derivative in time $O(|V||r|)$. According to Lemma 8.9, each derivative of an STE with cut border c is a union of at most $|r|$ many STEs with cut border at most c. Therefore, we can use algorithm $\mathcal{A}$ to also solve problem (2) in Lemma 8.4, by running it for each STE in the union separately. The smallest existing path in radix order can be found by taking the smallest returned path overall, for each STE in the union. To be precise, we need $O(f(n))$ time until the first output, and afterwards delay $O(|V|(|r| \cdot f(n) + |r| \cdot |V|^2))$. Since w is a prefix of lab(p), the algorithm needs to compute $w^{-1}L$ at most $|V|$ times in each of the $|V|$ iterations in line 7. □

If one is not interested in enumerating the simple paths in a particular order, then Lemma 8.11 and Lemma 4.17 immediately imply that EnumSimPaths($\mathcal{R}$) is in FPT delay for cuttable classes $\mathcal{R}$ of STEs. (The algorithm for Lemma 4.17 can output a witnessing path if it exists.) (By Remark 1, it is sufficient for the correctness of Yen's algorithm to be given simple paths. This observation propagates through Lemmas 8.4 and 8.11.) In the following, we will strengthen this to show that enumeration is even possible in radix order.

Enumeration in Radix Order. In the remainder, we will show how the decision algorithm for Theorem 3.5(a) can be adapted to return a smallest path in radix order. From now on, we refer to such a path as a *minimal* path. We show that Algorithm 2, for computing a simple path matching a 0-bordered STE, can be adjusted to compute a minimal path.

Lemma 8.12. *Let G be a graph, s and t nodes, and $r = A_1 \cdots A_{k_1} T^* A'_{k_2} \cdots A'_1$ a 0-bordered STE. If there exists a simple path from s to t matching r, then a shortest such path can be computed in time $2^{O(|r|)} \cdot |V|^3|E|$ and a minimal such path in time $2^{O(|r| \log |r|)} \cdot |V|^6|E|^2$.*

Proof. Since Algorithm 2 already solves the decision version of the problem, we only need to show that it can be adapted to compute a shortest, resp., minimal path in the required time. We first show that Algorithm 2 can output a shortest path. If Algorithm 2 returned 'yes', there exist nodes $x, y \in V$ and sets $X \in \hat{P}^{r_1}_{s,x}$ and $X' \in \hat{P}^{r_2}_{y,v}$, and a simple path p from x to y that matches T^* and is node disjoint from X and X' except for x and y. (See Lemma 4.12.) By definition of $P^{r_1}_{s,x}$, the nodes in $X \in \hat{P}^{r_1}_{s,x}$ form a path from u to x that matches $r_1 = A_1 \cdots A_{k_1}$. The construction of $\hat{P}^{r_1}_{s,x}$ in Lemma 4.10 allows us to order the elements in the sets such that they directly correspond to such a path. (In fact, the construction is analogous to [25, Lemma 5.2], which also shows that a witnessing path can be obtained.) So we can construct a path p_1 from u to x that uses only nodes in X and matches r_1 and a path p_2 from y to v that uses only nodes in X' and matches $r_2 = A'_{k_2} \cdots A'_1$. This also holds for a shortest such path, see Corollary 4.13.

To output a minimal path, we need to make some small changes to Algorithm 2. That is, we enumerate in line 2 all words $w_1 \in L(r_1)$ and compute $\hat{P}^{w_1}_{s,x} \subseteq^{k_1+k_2+1}_{\text{rep}} P^{w_1}_{s,x}$ for each such word. This way we can ensure that we really considered each word and, in particular, each prefix of a minimal simple path that matches r.[21] We proceed analogously in line 7 for all words $w_2 \in L(r_2)$.

Thus, we can use Algorithm 2 and iterate, for all words w_1 and w_2 and all nodes x, y over $\hat{P}^{w_1}_{s,x} \subseteq^{k+1}_{\text{rep}} P^{w_1}_{s,x}$ in line 2 and $\hat{P}^{w_2}_{y,t} \subseteq^{w_2}_{y,t}$ in line 8. Then we find a minimal simple path from x to y matching T^* in line 11 in time $O(|G|^2|r|^2|V|^2)$ with Proposition 4.2.

Concerning the time bounds, Algorithm 2 without changes has a running time of $2^{O(|r|)} \cdot |V|^{c+3}|E|$, see Lemma 4.15. Iterating over the words is in $O(|r|^{|r|})$ and using Proposition 4.2 instead of Lemma 4.1 and the reachability test for T^* adds a factor $O(|G||r|^2|V|^2)$. Rewriting $O(|r|^{|r|})$ into $2^{O(|r| \log |r|)}$ yields the result. □

Finally, the following result implies Theorem 8.7.

Lemma 8.13. *Let $\mathcal{R}$ be a class of STEs with cut border at most c. Then EnumSimPaths($\mathcal{R}$) is in FPT delay with radix order, to be more precise, with $2^{O(|r| \log |r|)} \cdot |V|^{c+6}|E|^2$ preprocessing time and delay $2^{O(|r| \log |r|)} \cdot |V|^{c+7}|E|^2$. If we only need order of increasing length, the preprocessing is $2^{O(|r|)} \cdot |V|^{c+3}|E|$ and the delay is $2^{O(|r|)} \cdot |V|^{c+4}|E|$.*

Proof. By Lemma 8.11, we only need to show the existence of an algorithm $\mathcal{A}$ that finds a minimal path within the required time bound. To this end, let $r \in \mathcal{R}$ and let c_1 and c_2 be the left and right cut border of r, respectively. Hence, $r = A_1 \dots A_{c_1} r' A'_{c_2} \cdots A'_1$. (If $c_i = 0$, then the respective part of r is simply missing.) We can compute, for all $u, v \in V$, all paths p_1 from s to u matching $A_1 \cdots A_{c_1}$ and all paths p_2 from v to t matching $A'_{c_2} \cdots A'_1$ in time $O(|V|^c)$.[22] We then do a loop over all pairs (p_1, p_2) of such paths that are node-disjoint. For each such pair, we will compute a *candidate path* $P_{(p_1,p_2)}$. The overall idea of the algorithm is that it first computes all such candidate paths and then, when it has iterated through all (p_1, p_2), takes the minimal one.

For the remainder of the proof, fix such a pair (p_1, p_2) and let p_{c_1} and p_{c_2} be the smallest paths (in radix order) obtained from p_1 and p_2 by considering the edge labels in G. The subexpression r' of r is of the form $r' = B'_{\text{pre}} T^* B'_{\text{suff}}$ and is 0-bordered. So we now search for a minimal simple path matching r' from u to v. We first delete in G all nodes in $(V(p_1) \setminus \{u\}) \cup (V(p_2) \setminus \{v\})$. Then, we perform a case distinction on the form of r'.

If $r' = A_1? \cdots A_{k_1}? T^* A'_{k_2}? \cdots A'_1?$, its language $L(r')$ is downward closed, so we can find a simple path p matching r' that is a minimal path using Proposition 4.2 and take $P_{(p_1,p_2)} = p_{c_1} p \, p_{c_2}$.

[21]If we start in Lemma 4.12 with a minimal simple path, we can replace P with a P' such that P' and R do not intersect. If additionally P and P' match the same word, the new path must also be a smallest one in radix order.

[22]For the purpose of the proof, it suffices to compute the paths without the edge labels here. We can find the labels on the edges in p_1 and p_2 that are smallest words in the corresponding expressions in radix order later.

For $r' = A_{c_1+1} \cdots A_{k_1} T^* A'_{k_2} \cdots A'_{c_2+1}$, we know from Lemma 8.12 that we can compute a minimal path p. We then define $P_{(p_1,p_2)} = p_{c_1} p \, p_{c_2}$.

If r' has another form, that is $r' = A_{c_1+1} \cdots A_{k_1} T^* A'_{k_2}? \cdots A'_1?$ or $r' = A_1? \cdots A_{k_1}? T^* A'_{k_2} \cdots A'_{c_2+1}$, we can also obtain a minimal simple path. In the first case, we again iterate over all words $w_1 \in A_{c_1+1} \cdots A_{k_1}$, compute the minimal path in $\hat{P}^{w_1}_{u,x} \subseteq^{k'_1+1}_{\text{rep}} P^{w_1}_{u,x}$, and use Proposition 4.2 to find a minimal path from x to t for the downward closed part. The other case is symmetric.

In each of the cases, the algorithm then iterates through all (p_1, p_2) and, for each such pair, adds a candidate path. Finally, it outputs the smallest candidate path.

Concerning the running time, we need time $O(|V|^c)$ to guess p_1 and p_2. To output a simple path (not necessarily minimal), we need $2^{O(|r|)} \cdot |V|^{c+3}|E|$ time, see Lemma 4.17. This is also the time we need to output shortest simple paths, since we can use the same algorithm. For minimal paths in radix order, we use Proposition 4.2 with running time $O(|G|^2|r|^2|V|^2)$ instead of Lemma 4.1 with running time $O(|G||r|)$ and instead of the reachability test for the T^* part. Furthermore, depending on r, we might need to enumerate all words $w_1 \in L(A_{c_1} \cdots A_{k_1})$ and $w_2 \in L(A'_{k_2} \cdots A'_1)$, and compute the rest of the algorithm depending on these words. Thus we need $2^{O(|r| \log |r|)} \cdot |V|^{c+6}|E|^2$ time overall in this case. The delay then follows from Lemma 8.11. □

8.3 Enumeration of Trails

The FPT result from Theorem 3.7 also carries over to enumeration problems. That is:

Theorem 8.14. *Let $\mathcal{R}$ be a class of STE that is almost conflict-free. Then, PEnumTrails($\mathcal{R}$) is in FPT delay, even when the paths need to be enumerated in radix order.*

More precisely, we obtain the following delays:

Lemma 8.15. *Let $\mathcal{R}$ be a class of STEs with at most c conflict positions. Then, PEnumTrails($\mathcal{R}$) is in FPT delay with radix order, to be more precise, in $2^{O(|r| \log |r|)} \cdot |E|^{c+11}$ preprocessing time and delay $2^{O(|r| \log |r|)} \cdot |E|^{c+12}$. If we only need order of increasing length, the preprocessing is $2^{O(|r|)} \cdot |E|^{c+6}$ and the delay is $2^{O(|r|)} \cdot |E|^{c+7}$.*

Proof. By Corollary 6.4, we have a bijection between the trails matching a word w in G and the simple paths matching $\sigma \cdot w$ in H, where H is obtained from G as in Lemma 6.3. Here, σ is an arbitrary label from Σ. Thus, we can use Lemma 8.11 on $H = (V_H, E_H)$ to enumerate the simple paths in the respective order and output the corresponding trails in G. We note that, due to Lemma 8.9, derivatives of STEs with at most c conflict positions again have at most c conflict positions. The computation time and size bounds can be found in Lemma 8.9.

So we need an algorithm that computes simple paths on H, matching $\sigma \cdot r$ and derivatives thereof in the respective order. Notice that the existence of such an algorithm is not immediate from our results on simple paths, since $\mathcal{R}$ is not necessarily cuttable. In fact, we need to relabel H and r as in (1)–(3) from the proof of Lemma 7.3. In (1), we relabeled r and some edges of H. Concerning the ordering of labels, we assume that, if $a < b$, then $a < \tilde{a} < b < \tilde{b}$. Notice that every A_i, T, or A'_i has only a or $\tilde{a}$ but not both, so this ordering does not affect the minimality of the path that we find. For every minimal path p matching $\sigma \cdot r$ in H there is a set S such that a minimal path in H_S matching $\sigma \cdot \tilde{r}$ will use the same nodes in the same order as p. We can compute, for each set S, a minimal path p_S in H_S, compare all such paths p_S, and take the minimal one. In (2), we only get rid of σ, so this will not change the minimality of a path. Finally, in (3), we use the same methods as in Lemma 4.16, which can be used to output simple paths in the respective order, see Lemma 8.13. Using the bijection between these simple paths and the trails in G, we can enumerate the trails.

So we can indeed output trails in the radix order or in order of increasing length. We now turn to the running time. Combining the blow-ups from the construction in Lemma 6.3 and the multiple graphs H_S we obtain from each different choice of S, we can find a shortest simple path that matches $\sigma \cdot \tilde{r}$ in time $2^{O(|r|)} \cdot |E|^{c+6}$ and a minimal simple path in time $2^{O(|r| \log |r|)} \cdot |E|^{c+11}$. Together with Lemma 8.11 this enables us to enumerate the simple paths and output the corresponding trails with delay $2^{O(|r|)} \cdot |E|^{c+7}$ for order of increasing length and delay $2^{O(|r|)} \cdot |E|^{c+12}$ for radix order. □

9 CONCLUSIONS

We have provided an extensive overview of evaluation and enumeration problems for regular path queries in graph databases under *arbitrary paths*, *shortest path*, *simple path*, and *trail* semantics. Our two main technical results are two dichotomies on the parameterized complexity of evaluating *simple transitive expressions (STEs)*, which are a class of regular expressions powerful enough to capture over 99.99% of the RPQs occurring in a recent practical study [15]. These dichotomies apply to *simple path* and *trail* semantics. Under simple path semantics, the central property that we require for a class of expressions so that evaluation is in FPT is *cuttability*, i.e., having bounded *cut borders* (also see Figure 2). Looking at Table 3, we see that the cut borders for expressions in practice are indeed very small: it is one for a^*b, two for abc^*, and zero in all other cases. Under trail semantics, the central property for evaluation in FPT is *almost conflict freeness*, i.e., a constant number of conflict positions. Looking again at the underlying data for Table 3, we discovered that all STEs had zero conflict positions. (We needed to look deeper again, because some classes in Table 3 aggregate others. For instance, "a^*b" also contains expressions of the form aa^*.)

Therefore, although evaluation under simple path and trail semantics of RPQs is known to be hard in general, it seems that the RPQs that users actually ask are much less complex.[23] In fact, since the vast majority (over 99%) of expressions in Table 3 has cut borders of at most two and no conflict positions, our FPT results in Theorems 3.5 and 3.7 imply that evaluation for this majority of expressions is in FPT with small parameter. Recall that, if P ≠ NP, this is impossible even for fixed expressions: evaluation for a^*ba^* or $(aa)^*$ under simple path semantics is NP-complete.

Beyond STEs. From a theoretical perspective it would be interesting to see to what extent our techniques can be used beyond STEs. We already observed that the FPT results extend to *unions of STEs*. Here, we briefly touch on another related class of expressions. Let an *extended STE* be a regular expression of the form

$$B_{\text{pre}} T_1^* \cdots T_k^* B_{\text{suff}},$$

where B_{pre} and B_{suff} are as in Definition 3.2.

Similarly to Section 3.5.1, if $B_{\text{pre}} = A_1 \cdots A_{k_1}$, we can define the left cut border to be the maximal i with $A_i \cap T_j \neq \emptyset$ for some $j \in \{1, \dots, k\}$; and zero if $B_{\text{pre}} = A_1? \cdots A_{k_1}?$ (analogously for the right cut border). With these definitions, the lower bound proof in Lemma 5.6 directly works for non-cuttable classes (that can be sampled) of these expressions and the FPT upper bound in Lemma 4.17 directly works for cuttable classes. Concerning trail semantics, it seems that the bounds do not immediately transfer and some more work is required. Another interesting direction would be to investigate to which extent the dichotomies extend to two-way regular path queries.

ACKNOWLEDGMENTS

We are grateful to Phokion Kolaitis for suggesting us to study enumeration problems on simple paths matching RPQs. We are also grateful to Holger Dell for pointing us to Theorem 4.6 and providing us with a proof sketch that the authors of [25] sent him. We acknowledge the many useful comments of the anonymous reviewers for ICDT 2018 and ACM TODS that helped us to significantly improve the structure and presentation of the article.

[23] A recent study confirmed this hypothesis on a corpus of 208M queries from Wikidata logs [14]. Here, about 39% of the unique queries used property paths.

REFERENCES

[1] Margareta Ackerman and Jeffrey Shallit. Efficient enumeration of words in regular languages. *Theoretical Computer Science (TCS)*, 410(37):3461–3470, 2009.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[3] Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In *International Conference on Management of Data (SIGMOD)*, pages 1421–1432, 2018.

[4] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5):68:1–68:40, 2017.

[5] Marcelo Arenas, Sebastián Conca, and Jorge Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *International Conference on World Wide Web (WWW)*, pages 629–638, 2012.

[6] Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. *CoRR*, abs/1212.6857, 2012. URL: http://arxiv.org/abs/1212.6857.

[7] Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. In *Symposium on Principles of Database Systems (PODS)*, pages 261–272, 2013.

[8] Pablo Barceló. Querying graph databases. In *Symposium on Principles of Database Systems (PODS)*, pages 175–188, 2013.

[9] Geert Jan Bex, Wim Martens, Frank Neven, and Thomas Schwentick. Expressiveness of XSDs: from practice to theory, there and back again. In *International Conference on World Wide Web (WWW)*, pages 712–721, 2005.

[10] Geert Jan Bex, Frank Neven, and Jan Van den Bussche. Dtds versus XML schema: A practical study. In *Proceedings of the Seventh International Workshop on the Web and Databases (WebDB)*, pages 79–84, 2004.

[11] Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and dtds. *ACM Trans. Database Syst.*, 35(2):11:1–11:47, 2010.

[12] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. Inferring XML schema definitions from XML data. In *International Conference on Very Large Data Bases*, pages 998–1009, 2007.

[13] Andreas Björklund, Thore Husfeldt, and Sanjeev Khanna. Approximating longest directed paths and cycles. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 222–233, 2004.

[14] Angela Bonifati, Wim Martens, and Thomas Tim. Navigating the maze of wikidata query logs. In *The Web Conference (WWW)*. ACM, 2019. To appear.

[15] Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *Proceedings of the VLDB Endowment (PVLDB)*, 11(2):149–161, 2017.

[16] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, October 1964.

[17] Yijia Chen and Jörg Flum. On parameterized path and chordless path problems. In *IEEE Conference on Computational Complexity*, pages 250–263. IEEE Computer Society, 2007.

[18] Mariano P. Consens and Alberto O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *Symposium on Principles of Database Systems (PODS)*, pages 404–416, 1990.

[19] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 323–330, 1987.

[20] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[21] Holger Dell. Personal communication, 2017.

[22] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[23] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: on completeness for W[1]. *Theoretical Computer Science (TCS)*, 141(1):109–131, 1995.

[24] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

[25] Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *Journal of the ACM*, 63(4):29:1–29:60, 2016.

[26] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science (TCS)*, 10(2):111–121, 1980.

[27] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Optimizing and parallelizing ranked enumeration. *Proceedings of the VLDB Endowment (PVLDB)*, 4(11):1028–1039, 2011.

[28] Martin Grohe and Magdalena Grüber. Parameterized approximability of the disjoint cycle problem. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 363–374, 2007.

[29] Oren Kalinsky, Yoav Etsion, and Benny Kimelfeld. Flexible caching in trie joins. In *International Conference on Extending Database Technology (EDBT)*, pages 282–293, 2017.

[30] Sampath Kannan, Z. Sweedyk, and Stephen R. Mahaney. Counting and random generation of strings in regular languages. In *Symposium on Discrete Algorithms (SODA)*, pages 551–557, 1995.
[31] Benny Kimelfeld and Yehoshua Sagiv. Extracting minimum-weight tree patterns from a schema with neighborhood constraints. In *International Conference on Database Theory (ICDT)*, pages 249–260, 2013.
[32] Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984.
[33] Andrea S. LaPaugh and Ronald L. Rivest. The subgraph homeomorphism problem. *Journal of Computer and System Sciences*, 20(2):133 – 149, 1980.
[34] Eugene L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.
[35] Leonid Libkin, Wim Martens, and Domagoj Vrgoč. Querying graphs with data. *Journal of the ACM*, 63(2):14:1–14:53, 2016.
[36] Katja Losemann and Wim Martens. The complexity of regular expressions and property paths in SPARQL. *ACM Transactions on Database Systems*, 38(4):24:1–24:39, 2013.
[37] Erkki Mäkinen. On lexicographic enumeration of regular and context-free languages. *Acta Cybernetica*, 13(1):55–62, 1997.
[38] Wim Martens, Frank Neven, Matthias Niewerth, and Thomas Schwentick. Bonxai: Combining the simplicity of DTD with the expressiveness of XML schema. *ACM Trans. Database Syst.*, 42(3):15:1–15:42, 2017.
[39] Wim Martens, Frank Neven, and Thomas Schwentick. Complexity of decision problems for simple regular expressions. In *Mathematical Foundations of Computer Science (MFCS)*, pages 889–900, 2004.
[40] Wim Martens and Tina Trautner. Evaluation and enumeration problems for regular path queries. In *International Conference on Database Theory (ICDT)*, pages 19:1–19:21, 2018.
[41] Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 12 1995.
[42] B. Monien. How to find long paths efficiently. In *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 239 – 254. North-Holland, 1985.
[43] Katta G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16(3):682–687, 1968.
[44] Neo4j. Intro to cypher. https://neo4j.com/developer/cypher-query-language/, 2017.
[45] Opencypher. www.opencypher.org. Visited on Sept. 14, 2017.
[46] Yehoshua Perl and Yossi Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *J. ACM*, 25(1):1–9, 1978.
[47] Stefan Plantikow, Mats Rydberg, and Petra Selmer. CIP2017-01-18 – configurable pattern matching semantics. https://github.com/boggle/openCypher/blob/isomatch/cip/1.accepted/CIP2017-01-18-configurable-pattern-matching-semantics.adoc. Visited on Aug. 08, 2017.
[48] Steven Skiena. *The Algorithm Design Manual (2. ed.)*. Springer, 2008.
[49] Aleksandrs Slivkins. Parameterized tractability of edge-disjoint paths on directed acyclic graphs. *SIAM Journal on Discrete Mathematics*, 24(1):146–157, 2010.
[50] Dimitri Surinx, George H. L. Fletcher, Marc Gyssens, Dirk Leinders, Jan Van den Bussche, Dirk Van Gucht, Stijn Vansummeren, and Yuqing Wu. Relative expressive power of navigational querying on graphs using transitive closure. *Logic Journal of the IGPL*, 23(5):759–788, 2015.
[51] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
[52] Domagoj Vrgoč. Personal communication, 2018. After a talk of Wim Martens at PUC Chile, the query was discussed in more detail and Domagoj Vrgoč, who attended the talk, informed us that he wrote the query.
[53] SPARQL 1.1 query language. https://www.w3.org/TR/sparql11-query/, 2013. World Wide Web Consortium.
[54] World wide web consortium. www.w3.org. Visited on Sept. 14, 2017.
[55] Jin Y. Yen. Finding the lengths of all shortest paths in n -node nonnegative-distance complete networks using 12n3 additions and n3 comparisons. *J. ACM*, 19:423–424, 07 1972.
[56] Mihalis Yannakakis. Graph-theoretic methods in database theory. In *Symposium on Principles of Database Systems (PODS)*, pages 230–242, 1990.
[57] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

Received August 2018; revised xx 2018; accepted xx 2019