THE COMPLEXITY OF REGULAR TRAIL AND SIMPLE PATH QUERIES ON UNDIRECTED GRAPHS

WIM MARTENS AND TINA POPP

University of Bayreuth, Germany

Abstract. We study the data complexity of regular trail and simple path queries on undirected graphs. Using techniques from structural graph theory, ranging from the graph minor theorem to group-labeled graphs, we are able to identify several tractable and intractable subclasses of the regular languages. In particular, we establish that trail evaluation for simple chain regular expressions, which are common in practice, is tractable, whereas simple path evaluation is tractable for a large subclass. The problem of fully classifying all regular languages is quite non-trivial, even on undirected graphs, since it subsumes an intriguing problem that has been open for 30 years.

1. Introduction

Graph databases are rapidly gaining importance [SBV+21]. Indeed, the graph query language GQL [DFG+21, GQLa] is now going through ISO standardization, which is a significant milestone. The standardization effort is a result of intense collaboration between industry and academia [GQLb]. While the GQL project is directly influenced \(^1\) by academic work on regular queries [RRV15] and GXPath [LMV16], it builds heavily on GCore [AAB+18], which is a light-weight query language for property graphs, developed by partners in academia and industry. In GQL [DFG+21], the evaluation of regular path queries (RPQs) is being considered in different modes: simple paths, trails, shortest paths, or at most \(k\) many paths. These variants are chosen to ensure that only a finite number of results needs to be returned.

The LDBC and the WG3 do not only collaborate on query language design. The LDBC also has a number of working groups focusing on the design of schema languages for property graphs. These groups have generated an initial proposal for key constraints for property graphs [ABD+21] and their generalization to cardinality constraints, which evolved into a deeper study on threshold queries [BDF+21]. Cardinality constraints are similar to key constraints in the sense that they do not impose uniqueness of an element in the database, but allow a given number to exist. Similarly, threshold queries return the answers of a given query until a given number of outputs is reached.

\textit{Key words and phrases:} Graph databases, regular path queries, regular languages, enumeration, query languages, trails, simple paths.

\(^*\) A preliminary version of this paper is accepted to PODS 2022 [MP22].

\(^1\) This can be seen in the GQL influence graph [GQLb].
Shortest paths and arbitrary path semantics have been studied in depth, for example in [Bar13, BFR19, BLLW12, BLR14, BOS15, BT16, CGLV02, CGLV00b, CGLV00a, CS21, DT01, FSS14, FGK+20, FLS98, MW95, LMV13, RRV17].

Although simple path evaluation has already been studied in the late 80’s and mid 90’s [CMW87, MW95], these early results showed that even relatively simple expressions are NP-hard to evaluate. After these results, research on RPQ evaluation mainly focused on unrestricted paths. A renewed interest in trails and simple paths [ACP12, BBG20, MNT20, MT19, LM13] was pushed by graph query language standards and systems, but is still lagging behind.

The present article continues the line of work that was started in [BBG20, MNT20]. These two papers fully classify the data complexity of RPQ simple path evaluation [BBG20] and trail evaluation [MNT20] over edge-labeled directed graphs. In a nutshell, these papers discovered the classes of regular languages $\text{SP}_{\text{tract}}$, resp., $\text{T}_{\text{tract}}$, for which, assuming $P \neq NP$, the data complexity of simple path, resp., trail evaluation, is tractable if and only if the expression belongs to $\text{SP}_{\text{tract}}$, resp., $\text{T}_{\text{tract}}$. For the broad community, the take-away messages are the following. First, the majority of RPQs that users ask in practice indeed belong to the tractable classes, which is good news. Second, $\text{SP}_{\text{tract}} \subset \text{T}_{\text{tract}}$, that is, “efficient” evaluation is possible for more RPQs under trail evaluation than under simple path evaluation.

This paper wants to take two navigational features of graph databases into account: undirected (aka bidirectional) edges and two-way navigation. Both of these features are supported by today’s leading graph database engines and query languages and significantly impact the trail and simple path evaluation of RPQs. Indeed, property graphs [AAB+17] model the data as a mixed multigraph, that is, a multigraph with directed and/or undirected edges and the GQL standardization [DFG+21] supports navigation on undirected edges.

Our focus here is on the data complexity of trail and simple path evaluation of ordinary RPQs on undirected graphs. As we discuss later, our results also contribute to understanding two-way RPQs (2RPQs).

**Related Work.** Cruz, Mendelzon and Wood [CMW87] designed one of the earliest navigational languages for graph databases. Motivated by early applications of graph databases, their language uses simple paths semantics, that is, they do not allow loops in the path. Mendelzon and Wood [MW95] observed that under simple path semantics querying a graph database is already NP-complete for relatively simple expressions like $a^*ba^*$ and $(aa)^*$. These two results heavily rely on the work of Fortune et al. [FHW80], who showed NP-completeness of the two disjoint paths problem on directed graphs, and LaPaugh and Papadimitriou [LP84], who showed that the even length simple path problem on directed graphs is NP-complete.

While Mendelzon and Wood [MW95] showed that the problem can be decided in polynomial time for downward closed languages, the overall complexity of simple path semantics was considered as too high and database systems therefore preferred arbitrary path semantics.

New interest in simple path semantics was sparked in 2010 when the W3C added regular expressions to SPARQL 1.1 queries in the form of SPARQL property paths. These property paths were evaluated under a semantics based on simple paths. Because of the studies about

---

2 We refer to [BMT20, BMT19] for detailed overviews of RPQs used in query logs.

3 Tigergraph allows undirected edges [Tig21, Defining a Graph Schema] and Neo4j Cypher allows specifying match patterns direction-agnostically. Two-way navigation is possible in either Tigergraph’s GSQL, Neo4j’s Cypher, and SPARQL 1.1.
the complexity of SPARQL 1.1 property paths [ACP12, LM13], SPARQL switched their semantics away from counting simple paths. Shortly afterwards, Bagan et al. [BBG20] provided a dichotomy for the data complexity of SimPath. They defined a class $SP_{tract}$ such that the problem is in P for each language in $SP_{tract}$ and NP-complete otherwise. Martens et al. [MNT20] give a similar dichotomy for paths that do not allow repetition of edges.

In the meantime, most work focused on arbitrary and shortest paths semantics, for which the evaluation problem is well-known to be tractable using standard product automata techniques [MW95]. Recently, Casel and Schmid [CS21] showed that this approach is essentially optimal. There is a wide literature on RPQs, which led to several surveys [AAB+$17, ARV19, Bar13, CGLV03, Woo12].

Indeed, RPQs have been extended in various ways. For example, to two-way regular path queries [BLLW12, BRV13, CGLV00a] that allow navigation in the reverse direction of an edge, or nested regular path queries [BPR12] that extend two-way regular path queries with additional node-tests. The favorable complexity of RPQs also carries over to these classes, see [Bar13] for an overview.

Modern graph query languages are often based on graph pattern matching. Examples are Cypher from Neo4j [FGG+$18], GSQL from TigerGraph [Tig21], and PGQL from Oracle [PGQ21], as well as industry/academia prototypes such as G-CORE [AAB+$18] and GPML [DFG+$21]. Conjunctions of RPQs (CRPQs) have been introduced in [CMW87] and been studied in several follow-up works, for example [BOS15, CGLV03, CGLV00a, CGLV02, DT01, Fig20, FGK+$20, FLS98, RRV17, RBV17]. Furthermore, CRPQs were extended with the ability to output and/or compare paths in [BLLW12, BBR14, BM17, FS13] and with data value comparisons [LMV16].

Graph theoreticians have extensively studied closely related problems on unlabeled or edge-colored undirected graphs, for example [ADF+$08, ADdLV+$10, APY91, GdLM12, GL+$09, GLMM13, GKM11, KT18, KW10, Men27, RS95]. We discuss the connection to this field in Section 2.6.

Our Contribution. Our contributions can be summarized as follows. After obtaining a number of closure and non-closure properties of the tractable classes of languages, we present a dichotomy on a generalization of the undirected two disjoint path problem with edge labels. We use this dichotomy to fully classify the complexity of trail and simple path evaluation of languages of the form $A^*wB^*$ on undirected graphs, where $A$ and $B$ are sets of symbols and $w$ is a word. We fully classify the complexity of languages of the form $w^*$, except in the case where this problem degenerates to testing the length of paths modulo some $k > 2$, which is an open problem since 1991 [APY91]. We study simple chain regular expressions (SCREs), which is a class of practically common languages and show that their trail evaluation is always tractable. This, however, is not the case for simple path evaluation, but we are able to identify large tractable subclasses. We generalize recent results on group-labeled graphs to obtain that the data complexity of parity languages is tractable for both trail and simple path evaluation. Finally, we show that all tractability results imply that enumeration of the output with polynomial-time delay is possible.

A preliminary version of this article is published as [MP22]. We extend this work by providing full proofs and showing that $SP_{tract} \subseteq UT_{tract}$.

While the last part focused on directed multigraphs, we now turn to undirected multigraphs. This will help us to understand the data complexity of RPQs on graph databases with undirected or bidirectional edges, which is supported by the major systems. Furthermore, it
We denote the empty word by $\varepsilon$. We use $\Sigma$, then we identify $r$-symbols in $t$ to abbreviate the $\Sigma$-symbols in $t$. We denote by $\wedge$ the size of a set. The $\Sigma$-language $L$ contains every word and their variants, like $\emptyset$, $\Sigma$, $\{\}$, $\{\varepsilon\}$, $\{\emptyset\}$, $\{\emptyset, \varepsilon\}$, etc. The size of a set of symbols $A$, denoted $|A|$, is the number of elements in the set. A word (over $\Sigma$) is a finite sequence $w = a_1 \cdots a_n$ of $\Sigma$-symbols. The length of $w$, denoted by $|w|$, is its number of symbols $n$. We denote the empty word by $\varepsilon$. The reverse of $w$ is $w^\text{rev} = a_n \cdots a_1$. For $0 \leq i \leq j \leq n$, we denote by $w[i,j]$ the substring $a_i \cdots a_j$ of $w$.

2. Definitions and Main Problems

We use $[n]$ to denote the set of integers $\{1, \ldots, n\}$. By $\Sigma$ we always denote a finite alphabet, that is, a finite set of symbols. A $\Sigma$-symbol is an element of $\Sigma$. We always denote symbols by $a$, $b$, $c$, $d$ and their variants, like $a'$, $a_1$, $b_1$, etc. We denote sets of symbols by uppercase letters like $A$, $B$, and their variants, like $A'$, $A_1$, $A'_1$ etc. The size of a set of symbols $A$, denoted $|A|$, is the number of elements in the set. A word (over $\Sigma$) is a finite sequence $w = a_1 \cdots a_n$ of $\Sigma$-symbols. The length of $w$, denoted by $|w|$, is its number of symbols $n$. We denote the empty word by $\varepsilon$. The reverse of $w$ is $w^\text{rev} = a_n \cdots a_1$. For $0 \leq i \leq j \leq n$, we denote by $w[i,j]$ the substring $a_i \cdots a_j$ of $w$.

2.1. Regular Expressions and RPQs. Regular expressions are defined as follows: $\emptyset$, $\varepsilon$, and every $\Sigma$-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 occurrences of $\Sigma$-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 thesis, we will blur the distinction between them (language acceptors vs. queries) and use “regular expression” and RPQ as synonyms.
The reversal of a language $L$ is $L^{rev} = \{w^{rev} \mid w \in L\}$. Given a language $L$ and a word $w$, the derivative\(^4\) of $L$ with respect to $w$ is defined as

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

2.2. Automata. A nondeterministic finite automaton (NFA) $N$ is a tuple $(Q, \Sigma, \delta, Q_I, Q_F)$, where $Q$ is a finite set of states, $\Sigma$ 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 $N$, denoted $|N|$, is its number of states $|Q|$. We define the language $L(N)$ of $N$ as usual.

We define an NFA $A$ to be a tuple $(Q, \Sigma, I, F, \delta)$ where $Q$ is the finite set of states; $I \subseteq Q$ is the set of initial states; $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation; and $F \subseteq Q$ is the set of accepting states.

2.3. Graph Databases and Paths. We use edge-labeled directed and undirected multigraphs as abstractions for graph databases. We start with the definition of directed multigraphs. An edge-labeled directed multigraph $G = (V, E, \mathcal{E})$ consists of a finite set of nodes $V$, a finite set of edges $E$, and a function $\mathcal{E} : E \rightarrow V \times \Sigma \times V$ that maps each edge identifier to a tuple $(v_1, a, v_2)$ describing the origin, the label, and the destination node of the edge. We denote $v_1$ by origin($e$), $a$ by lab($e$) and $v_2$ by destination($e$). We emphasize that $\mathcal{E}$ does not need to be injective, that is, there might be several edges with identical origin, label, and destination.

An edge-labeled undirected multigraph $G = (V, E, \mathcal{E})$ consists of a finite set of nodes $V$, a finite set of edges $E$, and a function $\mathcal{E} : E \rightarrow 2^V \cup \Sigma$ with $|\mathcal{E}(e) \cap \Sigma| = 1$ and $1 \leq |\mathcal{E}(e) \cap V| \leq 2$ for every $e \in E$. Given an edge $e \in E$, we denote by $\text{Node}(e) = \mathcal{E}(e) \cap V$ its nodes and by $\text{lab}(e) = \mathcal{E}(e) \cap \Sigma$ its label. For convenience, in undirected multigraphs, we also denote $\mathcal{E}(e)$ as $(u, a, v)$, where $(u, v) = \text{Node}(e)$ and $a = \text{lab}(e)$. As such, $(u, a, u)$ denotes a self-loop with label $a$ on node $u$. With this notion, $(u, a, v)$ and $(v, a, u)$ are the same in undirected graphs, and we will order $u$ and $v$ in our notation such that it optimizes readability.

Given an (un-)directed multigraph $G = (V, E, \mathcal{E})$, the size of $G$ is defined as $|V| + |E|$. A (simple) graph is a multigraph where $\mathcal{E}$ is injective. We sometimes denote $\mathcal{E}(e)$ as $(u, v)$ if the label does not matter.

For an undirected multigraph $G = (V, E, \mathcal{E})$ and a set $X \subseteq V$, the induced subgraph of $G$ on $X$ is the multigraph $G' = (X, E', \mathcal{E'}_{|E'})$ with $E' = \{e \mid e \in E \text{ and } \text{Node}(e) \subseteq X\}$. For a label $a$, we denote by $G_a$ the subgraph of $G = (V, E, \mathcal{E})$ restricted to edges labeled $a$, that is, $G_a = (V, E_a, \mathcal{E}_{|E_a})$ is a multigraph with $E_a = \{e \mid e \in E \text{ and } \text{lab}(e) = a\}$.

A path $p$ from $s$ to $t$ in an (un-)directed multigraph $G$ is a sequence of edges $e_1 \cdots e_k$ in $G$ such that $\mathcal{E}(e_1) \cdots \mathcal{E}(e_k)$ can be written as $(s, a_1, v_1)(v_1, a_2, v_2) \cdots (v_{k-1}, a_{k-1}, t)$ for some nodes $v_1, \ldots, v_{k-1} \in V$ and labels $a_1, \ldots, a_{k-1} \in \Sigma$. The set of nodes of path $p$ is $V(p) = \{s, v_1, \ldots, v_{k-1}, t\}$. The length of $p$, denoted by $|p|$, is the number of edges in $p$. A path is a trail if every edge $e$ appears at most once\(^5\) and a simple path if all its nodes are

\(^4\)Also known as Brzozowski derivative [Brz64].

\(^5\)We note that it is allowed that for $i \neq j$ it holds that $\mathcal{E}(e_i) = \mathcal{E}(e_j)$.
different, that is, if $|V(p)| = |p| + 1$. We note that each simple path is a trail but not vice versa.

A path $p$ from $v$ to $v$ is a simple cycle if $|V(p)| = |p|$. An example of a simple cycle is the path $p = e_1e_2$ with $\mathcal{E}(e_1) = (s,b,v_1)$, $\mathcal{E}(e_2) = (v_1,b,s)$ in Figure 1.

We denote $\text{lab}(e_1) \cdots \text{lab}(e_k)$ by $\text{lab}(p)$. Given a language $L \subseteq \Sigma^*$, path $p$ matches $L$ if $\text{lab}(p) \in L$. If $r$ is a regular expression (respectively $N$ is an NFA), we simplify notation and also say that $p$ matches $r$ when $p$ matches $L(r)$. We use $a$-edge to refer to an edge with label $a$ (that is, with $\text{lab}(c) = a$) and $a$-path to refer to a path that consists only of $a$-edges. Given a trail $p$ and two edges $e_1$ and $e_2$ in $p$, we denote the subpath of $p$ from $e_1$ to $e_2$ by $p[e_1,e_2]$. The concatenation of paths $p_1 = e_1 \cdots e_k$ and $p_2 = e_{k+1} \cdots e_n$ is simply the concatenation $p_1p_2$ of the two sequences. Notice that the last node of $p_1$ needs to be the same as the first node of $p_2$.

We illustrate some of these notions on the directed and undirected multigraphs in Figure 1. The path $p = e_1e_2$ with $\mathcal{E}(e_1) = (s,b,v_2)$, $\mathcal{E}(e_2) = (v_2,b,s)$ is a trail, both in the directed and the undirected multigraph, since $e_1 \neq e_2$. But it is no simple path because $|p| = 2$ and $V(p) = \{s,v_2\}$, and thus $|V(p)| \neq |p| + 1$. In both multigraphs there are two different $a$-paths from $s$ to $t$ that are simple paths. Both of them can be written as $(s,a,v_1)(v_1,a,v_2)(v_2,a,v_3)(v_3,a,t)$. Each simple path is also a trail. If we drop the restriction to simple paths, then there are infinitely many $a$-paths from $s$ to $t$ in the undirected multigraph, for example $(s,a,v_1)(v_1,a,s)(s,a,v_1)(v_1,a,v_2)(v_2,a,v_3)(v_3,a,t)$.

2.4. Main Problems. We will study variants of the SimPath and Trail problem.

<table>
<thead>
<tr>
<th>SimPath($L$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given: A directed multigraph $G = (V,E,\mathcal{E})$, two nodes $x, y \in V$</td>
</tr>
<tr>
<td>Question: Is there a simple path from $x$ to $y$ in $G$ that matches $L$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trail($L$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given: A directed multigraph $G = (V,E,\mathcal{E})$, two nodes $x, y \in V$.</td>
</tr>
<tr>
<td>Question: Is there a trail from $x$ to $y$ in $G$ that matches $L$?</td>
</tr>
</tbody>
</table>

Note that we study the data complexity of SimPath and Trail, that is, we assume that the language $L$ (the query) in the problems $\text{SimPath}(L)$ and $\text{Trail}(L)$ is not part of the input, but fixed. Therefore, each language gives rise to a different computational problem. We note that in this setting it plays no role whether $L$ is given as a DFA, NFA, or regular expression.

We will study these problems on undirected multigraphs. On directed (multi-)graphs these problems have already been studied in depth [BBG20, MNT20, MNP21]. We will denote variants on undirected multigraphs by adding a U.
2.5. Fundamental Subclasses of Regular Languages.

2.5.1. Downward Closed Languages (DC). A language $L$ is downward closed $^6$ (DC) if it is closed under taking subsequences. That is, for every word $w = a_1 \cdots a_n \in L$ and every sequence $0 < i_1 < \cdots < i_k < n + 1$ of integers, we have that $a_{i_1} \cdots a_{i_k} \in L$. Perhaps surprisingly, downward closed languages are always regular $^7$ [Hai69]. Furthermore, they can be defined by a clean class of regular expressions (which was shown by Jullien [Jul69] and later rediscovered by Abdulla et al. [ACBJ04]), which is defined as follows.

**Definition 2.1.** An atomic expression over $\Sigma$ is an expression of the form $(a + \varepsilon)$ or of the form $(a_1 + \cdots + a_n)^*$, where $a, a_1, \ldots, a_n \in \Sigma$. A product is a (possibly empty) concatenation $e_1 \cdots e_n$ of atomic expressions $e_1, \ldots, e_n$. A simple regular expression is of the form $p_1 + \cdots + p_n$, where $p_1, \ldots, p_n$ are products.

Another characterization is by Mendelzon and Wood [MW95], who show that a regular language $L$ is downward closed if and only if its minimal DFA $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$ exhibits the suffix language containment property, which says that if $\delta_L(q_1, a) = q_2$ for some symbol $a \in \Sigma$, then we have $L_{q_2} \subseteq L_{q_1}$.

**Theorem 2.2** ([ACBJ04, Hai69, Jul69, MW95]). The following are equivalent:

1. $L$ is a downward closed language.
2. $L$ is definable by a simple regular expression.
3. The minimal DFA of $L$ exhibits the suffix language containment property.

2.5.2. Tractable class for Regular Simple Path Queries ($\mathsf{SP}_{\text{tract}}$). Bagan et al. [BBG20] introduced $^8$ the class $\mathsf{SP}_{\text{tract}}$, which characterizes the class of regular languages $L$ for which the regular simple path query (SimPath) problem is tractable. While they studied SimPath on directed simple graphs, their results immediately carry over to multigraphs: since simple paths can use every node at most once, they cannot use more than one edge between any pair of nodes. Thus regarding simple path semantics, the results on a multigraph will be no different from the results on the underlying simple graph.

**Theorem 2.3** (Theorem 3 in Bagan et al. [BBG20]). Let $L$ be a regular language.

---

$^6$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.

$^7$They restrict $q_1, q_2$ to be on paths from $i_L$ to some state in $F_L$, but the property trivially holds for $q_2$ being a sink-state.

$^8$They called the class $\mathsf{C}_{\text{tract}}$, which stands for “tractable class”. We distinguish between $\mathsf{SP}_{\text{tract}}$ and $\mathsf{T}_{\text{tract}}$ here to avoid confusion between simple paths and trails.
(1) If $L$ is finite, then $\text{SimPath}(L) \in \mathcal{AC}^0$.

(2) If $L \in \text{SP}_{\text{tract}}$ and $L$ is infinite, then $\text{SimPath}(L)$ is $\text{NL}$-complete.

(3) If $L \notin \text{SP}_{\text{tract}}$, then $\text{SimPath}(L)$ is $\text{NP}$-complete.

One characterization of $\text{SP}_{\text{tract}}$ is the following (Theorem 6 in [BBG20]):

**Definition 2.4.** $\text{SP}_{\text{tract}}$ is the set of regular languages $L$ such that there exists an $i \in \mathbb{N}$ for which the following holds: for all $w_l, w, w_r \in \Sigma^*$ and $w_1, w_2 \in \Sigma^+$ we have that, if $w_l w_1^i w_2^i w_r \in L$, then $w_l w_1^i w_2^i w_r \in L$.

Bagan et al. [BBG20] also gave a characterization of $\text{SP}_{\text{tract}}$ in terms of regular expression:

**Theorem 2.5.** Let $L$ be a regular language. Then $L$ belongs to $\text{SP}_{\text{tract}}$ if and only if $L$ can be written as a union of regular expressions of the form

$$w_l(w_1 + \varepsilon)(A_1^{\geq k_1} + \varepsilon)(w_2 + \varepsilon) \cdots (A_n^{\geq k_n} + \varepsilon)w_r$$

for some $n, k_1, \ldots, k_n \in \mathbb{N}$, words $w_l, w_1, \ldots, w_n, w_r \in \Sigma^*$, and sets $A_1, \ldots, A_n \subseteq \Sigma$.

2.5.3. **Tractable class for Regular Trail Queries ($T_{\text{tract}}$).** Martens et al. [MNT20] introduced the class $T_{\text{tract}}$, which characterized the class of regular language $L$ for which the regular trail query (Trail) problem is tractable. They show in an extended version [MNP21] that their results also hold on multigraphs.

**Theorem 2.6** (Theorem 4.1 in Martens et al. [MNT20, MNP21]). Let $L$ be a regular language.

(1) If $L$ is finite, then $\text{Trail}(L) \in \mathcal{AC}^0$.

(2) If $L \in T_{\text{tract}}$ and $L$ is infinite, then $\text{Trail}(L)$ is $\text{NL}$-complete.

(3) If $L \notin T_{\text{tract}}$, then $\text{Trail}(L)$ is $\text{NP}$-complete.

Martens et al. [MNT20] give several equivalent definitions of $T_{\text{tract}}$, one of them is the following:

**Definition 2.7.** $T_{\text{tract}}$ is the class of regular languages $L$ such that there exists an $i \in \mathbb{N}$ for which the following holds: for all $w_l, w, w_r \in \Sigma^*$ and $w_1 = aw_1'$ and $w_2 = w_2'$ we have that $w_l w_1 w w_2 w_r \in L$ implies $w_l w_1 w_2 w_r \in L$.

2.6. **Context.** The task of understanding $\text{USimPath}$ and $\text{UTrail}$ for all regular languages (which is a major step towards understanding $\text{SimPath}$ and $\text{Trail}$ for all two-way regular languages) is very general and subsumes a long open standing problem to which we will get later in this section. First, consider the following problems.

<table>
<thead>
<tr>
<th>$k\text{DisjointPaths}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> A multigraph $G$, node pairs $(s_1, t_1), \ldots, (s_k, t_k)$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Are there pairwise disjoint paths from $s_i$ to $t_i$ in $G$ for every $i \in [k]$?</td>
</tr>
</tbody>
</table>

These problems come in four variants for each $k$: for multigraphs that are directed or undirected, and for paths that are required to be node-disjoint (no common node) or edge-disjoint (no common edge).

<table>
<thead>
<tr>
<th>Mod-$k$-Path</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> A multigraph $G$, nodes $s$, $t$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is there a path of length 0 modulo $k$ from $s$ to $t$ in $G$?</td>
</tr>
</tbody>
</table>
These problems also come in four variants: for multigraphs that are directed or undirected, and for simple paths or trails.

These problems are very relevant to SimPath and Trail on directed and undirected multigraphs: the Mod-$k$-Path problem is equivalent to deciding if there is a simple path/trail from $s$ to $t$ that matches $(a^k)^*$. Let $L_{\text{equiv}} = a^*a_1a_2a_2a_3\ldots a_{k-1}a_{2k-2}a^*$, where $a, a_1, \ldots, a_{2k-2} \in \Sigma$ are pairwise different. The $k\text{DisjointPaths}$ problem is equivalent to deciding if there is a simple path/trail from $s_1$ to $t_k$ that matches $L_{\text{equiv}}$ in a $G$.\footnote{We use two different symbols $a_i a_{k-1+i}$ to simulate a directed edge even if $G$ is an undirected multigraph. If $G$ is a directed multigraph, the language can be simplified.} We make this equivalence more explicit.

- We can find a simple path/trail from $s$ to $t$ in $G$ that matches $L_{\text{equiv}}$ as follows: We iterate over all tuples $(p_1, \ldots, p_{k-1})$ of node/edge-disjoint simple paths/trails $p_i$ such that $p_i$ matches $a_i a_{k-1-i}$. Assume that path $p_i$ is from $u_i$ to $v_i$. It then remains to test for $k$ node/edge-disjoint paths matching $a^*$ from $u_1$, $v_1$, $u_{i+1}$, and from $v_{k-1}$ to $t$. This is equivalent to solving $k\text{DisjointPaths}$ in the subgraph $G_u$ of $G$.

- We can solve $k\text{DisjointPaths}$ as follows: (Re)label every edge in $G$ with $a$. Add new nodes and edges labeled $a_i a_{k-1+i}$ from $t_i$ to $s_{i+1}$ for each $i \in [k-1]$. Then this is equivalent to deciding if there is a simple path/trail from $s_1$ to $t_k$ in $G$ that matches $L_{\text{equiv}}$.

Since Mod-$k$-Path and $k\text{DisjointPaths}$ are NP-complete for $k \geq 2$ on directed graphs \cite{LP84,FHW80}, these problems can be used to show NP-hardness. For example, Mendelzon and Wood \cite{MW95} use TwoDisjointPaths to show that SimPath$(a^*ba^*)$ is NP-complete. Bagan et al. \cite{BBG20} also use a reduction from TwoDisjointPaths to show that SimPath$(L)$ is NP-complete for regular languages $L \notin \Sigma_{\text{tract}}$, as do Martens et al. \cite{MNT20} for trail semantics.

On the other hand, some of these problems are tractable on undirected multigraphs. This allows us to prove tractability results for USimPath and UTrail by reductions to cases in which Mod-$k$-Path and $k\text{DisjointPaths}$ are tractable.

We now discuss in detail what is known about Mod-$k$-Path and $k\text{DisjointPaths}$ on undirected (multi-)graphs.

\textbf{Unlabeled, Undirected.} The famous minor theorem \cite{RS95} implies that $k\text{DisjointPaths}$ is tractable for every fixed $k$ on undirected graphs, independent of whether we require node-disjoint or edge-disjoint paths. Indeed, for node-disjoint paths, this problem is equivalent to deciding if the set of $k$ distinct edges $(s_1, t_1), \ldots, (s_k, t_k)$ is a minor of a given undirected graph. We note that for node-disjoint paths, it does not play a role whether we consider multigraphs or restrict ourselves to graphs, since every node can only be used once and therefore, no edge between a pair of nodes can be used more than once. For edge-disjoint paths, Jarry and Pérennes \cite[Lemma 2]{JP09} show how to decide if $k$ edge-disjoint paths in undirected, unlabeled multigraphs in polynomial time exist: They split each edge before applying the line graph construction and then use the minor theorem.

Therefore, the following Proposition follows from Robertson and Seymour’s Graph Minor Project \cite{RS95} and Jarry and Pérennes \cite[Lemma 2]{JP09}:

\textbf{Proposition 2.8.} $k\text{DisjointPaths}$ on undirected multigraphs is in polynomial time for node- and edge-disjoint paths.
The **Mod-k-Path** problem for \( k = 2 \) is tractable for simple paths [LP84].\(^{10}\) The situation for \( k \geq 3 \) is rather intriguing. Arkin et al. [APY91] proved that, for every fixed \( k > 1 \), one can test in polynomial time whether there is an undirected simple path of length **different from** \( 0 \mod k \) between two given nodes. Although this result allows to solve **Mod-k-Path** for \( k = 2 \), there is no clear reduction from **Mod-k-Path** to this problem if \( k > 2 \). Indeed, the complexity of **Mod-k-Path** for \( k = 3 \) has been open for 30 years [APY91].

When it comes to parity (mod 2) conditions, there has been recent progress. For instance, the minor theorem has been extended to incorporate parities [KRW11, Huy09], which was a non-trivial effort. As a consequence, we now know that we can test in polynomial time if a given undirected graph has, \( k \) node-disjoint simple paths of even length.

Concerning edge-disjoint trails, Kawarabayashi and Kobayashi [KK16] defined the **extended line graph** construction, which maintains parity information and allows to transfer from known results on node-disjoint simple paths with parity constraints. The construction replaces every vertex with a clique in which every edge is subdivided into two edges. The size of the clique replacing \( v \) is the number of edges adjacent to \( v \).

We show how this idea can be tweaked to cope with arbitrary modulors. If we want to test path lengths modulo \( m \), we subdivide every edge of the new cliques into \( m \) edges. Furthermore, start- and end-nodes \( (s_1, t_1), \ldots, (s_k, t_k) \) can be incorporated by adding extra nodes.

**Lemma 2.9.** Let \( G \) be an undirected multigraph. There are trails from \( s_i \) to \( t_i \) of length \( j_i \mod m_i \) in \( G \) which are pairwise edge-disjoint if and only if there are simple paths from \( s_i^* \) to \( t_i^* \) of length \( j_i \mod m_i \) in the extended line-graph of \( G \) with respect to \( (s_i, t_i)_{i \in [k]} \) modulo \( m_1 \cdot m_2 \cdots m_k \) which are pairwise node-disjoint.

**Proof.** We start with the exact definition of extension to the extended line graph from Kawarabayashi and Kobayashi [KK16] which now incorporates node-pairs and arbitrary modulo \( m \) conditions. Let an undirected multigraph \( G = (V, E, \mathcal{E}) \), and node-pairs \( (s_1, t_1), \ldots, (s_k, t_k) \) be given. Let \( < \) be some order on \( E \). We can assume without loss of generality that \( s_i \neq t_i \) or \( j_i \neq 0 \) for each \( i \) (otherwise, this pair is trivially satisfied and can thus be removed). The extended line graph

\[
L_{m_1 \cdot m_2 \cdots m_k}(G, (s_j, t_j)_{j \in [k]}) = ((G^*), (s_j^*, t_j^*)_{j \in [k]})
\]

where \( G^* = (V^*, E^*, \mathcal{E}^*) \) is an undirected graph defined by:

\[
\begin{align*}
V^* &= V_1^* \cup V_2^* \cup \ldots \cup V_t^* \\
V_1^* &= \{(v, e) \mid v \in V, e \in E, e \text{ is adjacent to } v\} \\
V_2^* &= \{(v, \{e_1, e_2\}, i) \mid v \in V, e_1, e_2 \in E, e_1 \text{ and } e_2 \text{ are adjacent to } v \text{ and } i \in [m - 1]\} \\
V_s^* &= \{s_j^* \mid j \in [k]\} \cup \{(s_j^*, (s_j, e), i) \mid e \text{ is adjacent to } s_j, j \in [k], i \in [m - 1]\} \\
V_t^* &= \{t_j^* \mid j \in [k]\} \cup \{(t_j^*, (t_j, e), i) \mid e \text{ is adjacent to } t_j, j \in [k], i \in [m - 1]\} \\
E^* &= E_1^* \cup E_2^* \cup \ldots \cup E_t^* \\
E_1^* &= \{((v, e_1), (v, \{e_1, e_2\}, 1)), ((v, \{e_1, e_2\}, m - 1), (v, e_2)) \mid e_1 < e_2, (v, e_1), (v, \{e_1, e_2\}) \in V^*, v \in V\} \\
&\quad \cup \{(v, \{e_1, e_2\}, i)(v, \{e_1, e_2\}, i + 1) \mid (v, \{e_1, e_2\}, i), (v, \{e_1, e_2\}, i + 1) \in V^*, i \in [m - 2]\}
\end{align*}
\]

\(^{10}\)They attribute the first algorithm for this problem to Jack Edmonds due to private communication.
$E_2 = \{((v_1, e)(v_2, e)) | e$ is an edge connecting $v_1$ and $v_2\}$

$E_s = \{((s_j^*, (s_j^*, (s_j, e), 1)), ((s_j^*, (s_j, e), m-1), (s_j, e)) |$

$s_j^* (s_j, e), 1), (s_j, e) \in V^*, j \in [k]\}$

$\cup \{((s_j^*, (s_j, e), i), (s_j^*, (s_j, e), i + 1)) |$

$(s_j^*, (s_j, e), i), (s_j^*, (s_j, e), i + 1) \in V^*, j \in [k], i \in [m-2]\}$

$E_t = \{(t_j^*, (t_j^*, (t_j, e), 1)), ((t_j^*, (t_j, e), m-1), (t_j, e)) |$

$t_j^* (t_j, e), 1), (t_j, e) \in V^*, i \in [k]\}$

$\cup \{((t_j^*, (t_j, e), i), (t_j^*, (t_j, e), i + 1)) |$

$(t_j^*, (t_j, e), i), (t_j^*, (t_j, e), i + 1) \in V^*, j \in [k], i \in [m-2]\}$

Furthermore, $E^*(x, y) = (x, y)$ for all $(x, y) \in E^*$. We note that the sets $V^*_s \cup V^*_t$ and $E^*_s \cup E^*_t$ are empty if no distinguished nodes $s_j, t_j$ exist.

We now prove the lemma. Let $((G^*), \bigcup_{j=1}^k E^*_j) = L_{m_1, m_2, \ldots, m_k} (G, (s_j, t_j)_{j \in [k]})$. Since in $G^*$ each path through each newly added clique has length 0 mod $m_i$, only edges of the form $((v_1, e)(v_2, e))$ count towards length modulo $m_j$. Given a simple path from $s_j^*$ to $t_j^*$ in $G^*$, one can construct a path from $s_j$ to $t_j$ with the same length modulo $m_j$ by replacing edges of the form $((v_1, e)(v_2, e))$ with $e$ and omitting all edges of different forms. On the other hand, starting from a trail in $G$, one can add transitions (via the cliques) between each pair of edges to obtain a simple path in $G^*$. Since adding those transitions does not count towards the length modulo $m_j$, the so-constructed simple path has the same length modulo $m_j$. Finally, we note that node-disjoint paths in $G^*$ cannot share edges of the form $((v_1, e)(v_2, e))$, thus their corresponding trails must be edge-disjoint and vice versa.

This version of the extended line graph will be useful in Section 11.

**Labeled, Undirected.** On undirected labeled graphs, the problem USimPath$(ab)^*$ has been studied under the name *properly edge-colored (PEC) simple path* in a two-colored graph. Here, a path is defined to be PEC if its adjacent edges have different colors. It is decidable polynomial time if a PEC simple path from $s$ to $t$ exists. For two colors, this result is attributed to Edmonds [Man95] and was generalized by Szeder [Sze03] to any number of colors. Abouelaoualim et al. [ADF+08] give a polynomial time that decides in polynomial time if a PEC trail exists and also works on multigraphs. Thus USimPath$(ab)^*$ and UTrail$(ab)^*$ are in polynomial time.

Next, we discuss a number of results on two disjoint paths, since we will sometimes rely on them in the paper. To this end, for two languages $L_1$ and $L_2$, the problem of finding *node-* (respectively, *edge-*) *disjoint* $L_1/L_2$ *simple paths* (respectively, *trails*) refers to finding two node- (respectively, edge-) disjoint simple paths (respectively, trails)

$p_1$ and $p_2$ such that $lab(p_1) \in L_1$ and $lab(p_2) \in L_2$ between given nodes $(s_1, t_1), (s_2, t_2)$.

As such, finding disjoint $a^*/b^*$ paths is equivalent to finding two monochromatic disjoint paths in an undirected graph with two edge colors. For edge-disjointness, the latter problem is in P as $a$-paths will always be edge-disjoint from $b$-paths. The problem therefore reduces to reachability. For node-disjointness, the problem is NP-complete, see [GdLMM12, Theorem 16]. Finding node- or edge-disjoint $(ab)^*/(ab)^*$ paths is NP-hard [ADF+08] (a closely

\[\text{The relationship between such problems and ours is that finding node-disjoint } L_1/L_2 \text{ simple paths is closely related to finding a single simple path labeled } L_1aL_2, \text{ for some label } a \text{ (similar for edge-disjoint paths and trails).}\]
related problem was studied in [CMM⁺94]). Finally, node-disjoint \((ab)^*/a^*\) simple paths is NP-complete by Gourvès et al. [GdLMM12, Proof of Corollary 10].

2.7. Related Work. Paths on undirected graphs have been extensively studied. A seminal line of work was accomplished by Robertson and Seymour [RS95] and resulted in the Graph Minor Theorem, which was followed by work on simplifying the proof, extending the results, and obtaining better running times [GKMW11, KTW18, KW10]. Huynh [Huy09] extended this minor theorem to group-labelled graphs. Although group-labelled graphs are quite different form undirected graphs, they coincide modulo 2 (but not for other modulos). Thus, in particular, Huynh’s work of implies a minor theorem with parity conditions, i.e., it gives an algorithm to find structures with certain path length modulo 2 in polynomial time. Independently and with different methods, Kawarabayashi et al. [KRW11] also proved this result and with an improved running time.

The problem of finding a simple path of length modulo 3 has also been studied extensively, but is not yet solved on undirected graphs. Arkin et al. [APY91] give a linear-time algorithm to decide whether all paths between two specified nodes are of length \(P \mod Q\), for fixed integers \(P\) and \(Q\).

Many works also look for paths of certain modulo in very restricted kinds of graphs. For example, Deng and Papadimitriou [DP91] show that between any two nodes of a cubic, planar, three-connected graph there are three paths whose lengths are 0, 1, and 2 modulo 3, respectively. Amar and Manoussakis [AM90] give several sufficient conditions on the half-degrees of a bipartite digraph for the existence of cycles and paths of various lengths.

Another line of work, which is similar to what we consider here, was obtained by Abouelaoualim et al. [ADF⁺08, ADdlV⁺10] and Gourvès et al. [GdLMM12, GLM⁺09, GLMM13]. They study simple paths and trails with certain color conditions in edge-colored graphs. Also in this setting, people studied this problem on restricted variants of graph. For example, Manoussakis [Man95] studied the existence of properly edge colored paths in complete graphs, while Abouelaoualim et al. [ADdlV⁺10] focused on graphs with degree conditions. Others try to characterize graphs that can be colored with 2 colors such that there are properly edge-colored (simple) paths or trails between any pair of nodes, see e.g. [GM18].

3. First Observations

Let \(\text{USP}_{\text{tract}}\) be the class of regular languages for which \(\text{USimPath}\) is in P and let \(\text{UT}_{\text{tract}}\) be the corresponding class for UTrail. While \(\text{SP}_{\text{tract}}\) and \(\text{T}_{\text{tract}}\) are closed under intersection and union [BBG20, MNT20], \(\text{USP}_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are not closed under intersection if \(P \neq \text{NP}\).

**Theorem 3.1.** The following hold if \(P \neq \text{NP}\).

(a) \(\text{USP}_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are closed under (finite) union.
(b) \(\text{USP}_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are not closed under intersection.
(c) \(\text{USP}_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are not closed under complement.
(d) \(\text{USP}_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are closed under taking derivatives, that is, if \(L\) is tractable, then so is \(w^{-1}L = \{u \mid wu \in L\}\).
(e) \(\text{USP}_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are closed under reversal.
Theorem 3.2.

(a) \(\text{SP}_{\text{tract}} \subseteq \text{USP}_{\text{tract}}\).
(b) \(\text{SP}_{\text{tract}} \subseteq \text{UT}_{\text{tract}}\).
(c) If \(P \neq \text{NP}\), then \(T_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are incomparable.

Proof. We first prove (a). If we have two languages \(L_1, L_2\) for which \(\text{USimPath}(L_1)\) and \(\text{USimPath}(L_2)\) (\(\text{UTrail}(L_1)\) and \(\text{UTrail}(L_2)\), respectively) are in P, we obtain a P algorithm for \(\text{USimPath}(L_1 \cup L_2)\) (\(\text{UTrail}(L_1 \cup L_2)\), respectively) by using the two P algorithms for \(L_1\) and \(L_2\). If any of them answers “yes”, there is a simple path (trail, respectively) matching \(L_1 \cup L_2\).

We start the proof of (b) with \(\text{USP}_{\text{tract}}\). By Theorem 11.1 for the language containing an even number of \(a\)'s and arbitrary number of \(b\)'s \(\text{USimPath}\) is tractable, that is \(\text{USimPath}(b^*(ab*ab*)^*)\) is in P. Since \(a^*b^*\) is downwardclosed, \(\text{USimPath}(a^*b^*)\) is in P. On the other hand, \(\text{USimPath}((aa)^*b^*)\) is NP-hard. NP hardness follows from \(G_{3\text{SAT}}\) with words \(w_s = a, w_b = a, w_m = a, w_r = b, w_o = w_t = \varepsilon\) and Theorem 4.1.

We now turn to \(\text{UT}_{\text{tract}}\). By Theorem 11.1 the language containing an even number of \(a\)'s and arbitrary number of \(b, c,\) and \(d\) is tractable for \(\text{UTrail}\), that is \(\text{UTrail}((b + c + d)^*(a(b + c + d)^*)^*a(b + c + d)^*)^*\) is in P. Since \(a^*(b + c)^*a^*(b + d)^*\) is downwardclosed, \(\text{UTrail}(a^*(b + c)^*a^*(b + d)^*)\) is in P. Let \(L\) be the intersection of both languages, that is, \(L = \{a^n(b + c)^*a^m(b + d)^* | n + m \text{ being even}\}\). We show that \(\text{UTrail}(L)\) is NP-hard. NP hardness follows from \(G_{3\text{SAT}}\) with words \(w_s = a, w_b = a, w_m = a, w_o = b, w_r = d, w_t = \varepsilon\) and Theorem 4.1.

For (c), we first observe that by (a), \(\text{USP}_{\text{tract}}\) and \(\text{UT}_{\text{tract}}\) are closed under union. If they were also closed under complement, we could simulate closure under intersection, which would contradict (b).

We now turn to prove (d). Let \(w\) be an arbitrary word. We first prove that there is a word \(w'\) of constant length such that \(w^{-1}L = (w')^{-1}L\). Let \(L\) be a regular language. Let \(A\) an DFA for the language \(L\), and \(w\) an arbitrary word. Then a DFA \(A'\) for \(w^{-1}L\) can be obtained by changing the starting state of \(A\). (If \(w^{-1}L = \emptyset\), we can choose a sink state as start.) Let \(q\) be the start state of \(A\) and \(q'\) be the start state of \(A'\). Now we can find a word \(w'\) of length at most \(|A|\) such that \(\delta^*(q, w') = q'\).

We are now ready to prove (d). Let a graph \(G\) with nodes \(s\) and \(t\) be given. We can find a simple path (or trail) from \(s\) to \(t\) matching \(w^{-1}L\) as follows: we add a new node \(s'\) and a path labeled \(w'\) from \(s'\) to \(s\). Then there exists a simple path (trail, respectively) from \(s\) to \(t\) matching \(L'\) if and only if there exists a simple path (respectively trail) from \(s'\) to \(t\) matching \(L\). Thus, if \(L \in \text{USP}_{\text{tract}}\) (in \(\text{UT}_{\text{tract}}\), respectively), it follows that \(L' \in \text{USP}_{\text{tract}}\) (in \(\text{UT}_{\text{tract}}\), respectively).

Part (e) is trivial because the question concerns undirected graphs.

Although \(\text{USimPath}\) and \(\text{UTrail}\) are tractable for every language for which \(\text{SimPath}\) is tractable, \(\text{UTrail}\) and \(\text{Trail}\) are incomparable. An intuitive reason is that a trail for the language \((abc)^*\) in directed multigraphs is easy to find since loops can always be removed. On the other hand, we cannot use the same argument on undirected multigraphs since every edge can be used in one or the other direction and we can only remove loops if the joint edge is used in the same direction.
directed edges $e_1, e_2$ such that if $\mathcal{E}(e) = (u, a, v)$, then $\mathcal{E}'(e_1) = (u, a, v)$ and $\mathcal{E}'(e_2) = (v, a, u)$. Since a simple path can use at most one edge between any pair of nodes, a simple path in $G'$ can use either $e_1$ or $e_2$, but not both. Thus, there is a simple path from $s$ to $t$ that matches $L$ in $G$ if and only if there is a simple path from $s$ to $t$ that matches $L$ in $G'$.

We now prove (b). We will use that Bagan et al. [BBG20, Theorem 6], see Theorem 2.5, give a definition of $\text{SP}_{\text{tract}}$ in terms of regular expressions, showing that a language is in $\text{SP}_{\text{tract}}$ if and only if it can be expressed as a union of regular expressions of the form

$$w_1(A_1^{>k_1} + \varepsilon)(w_2 + \varepsilon)(A_2^{>k_2} + \varepsilon)\cdots(w_n + \varepsilon)(A_n^{>k_n} + \varepsilon)w_{n+1}$$

for some $n \in \mathbb{N}$, words $w_j \in \Sigma^*$ with $j \in [n+1]$, sets $A_i \subseteq \Sigma$ and numbers $k_i \in \mathbb{N}$ with $i \in [n]$. Since $\text{UT}_{\text{tract}}$ is closed under union by Theorem 3.1, it suffices to prove that $\text{UT}_{\text{tract}}(r)$ is tractable for each regular expression $r$ of the form $(\phi)$.

Let $G = (V, E, \mathcal{E})$ be an undirected multigraph, $s, t \in V$, and $r$ of the form $(\phi)$. We will construct in polynomial time a directed graph $G'$ and regular expression $r'$ such that there exists a trail matching $r$ from $s$ to $t$ in $G$ if and only if there is a trail from $s$ to $t$ matching $r'$ in $G'$ that satisfies some additional restrictions. We then show that its existence can be tested in polynomial time.

Let $\$_1, \$_2$ be two symbols which occurs neither in $G$ nor in $r$. We construct from $G = (V, E, \mathcal{E})$ a new directed graph $G' = (V', E')$ with $V' = V \cup \{x_e, y_e \mid e \in E\}$, and $E' = \{(u, \$_1, x_e), (v, \$_2, x_e), (x_e, a, y_e), (y_e, \$_2, u), (\$_1, v) \mid e \in E, \mathcal{E}(e) = (u, a, v)\}$. Intuitively, we replace every edge $e$ with the gadget presented in Figure 2. We note that these gadgets introduce loops labeled $\$_1a\$_1$ from $u$ to $u$ and labeled $\$_2a\$_2$ from $v$ to $v$.

![Figure 2](image-url)

Let $h: \Sigma \to \Sigma \cup \{\$_1, \$_2\}$ be a substitution with $h(\sigma) = (\$_1\sigma\$_2 + \$_2\sigma\$_1)$ for each $\sigma \in \Sigma$. That is, if $\sigma = a_1a_2\cdots a_n \in \Sigma^+$, then $h(\sigma)$ is a set of words, namely, $h(\sigma) = (\$_1a_1\$_2 + \$_2a_1\$_1)(\$_1a_2\$_2 + \$_2a_2\$_1)\cdots(\$_1a_n\$_2 + \$_2a_n\$_1)$, furthermore, $h(\varepsilon) = \varepsilon$, and $h(A) = (\$_1A\$_2 + \$_2A\$_1)$ for every set $A$. Depending on $r$, we define

$$\tilde{r} = h(w_1)((h(A_1))^{\geq k_1} + \varepsilon)(h(w_2) + \varepsilon)((h(A_2))^{\geq k_2} + \varepsilon)\cdots(h(w_n) + \varepsilon)((h(A_n))^{\geq k_n} + \varepsilon)h(w_{n+1}).$$

Note that $n, w_j$ for all $j \in [n+1]$, and $A_i, k_i$ for all $i \in [n]$ are defined by $r$.

We now prove that there is a trail $p$ matching $r$ from $s$ to $t$ in $G$ if and only if there is a trail $p'$ matching $\tilde{r}$ in $G'$. Let $p = e_1 \cdots e_n$ be a trail from $s$ to $t$ in $G$ that matches $r$. Then a trail $p'$ can be obtained from $p$ by replacing every edge $e_i$ with its corresponding path matching $\$_1\text{lab}(e_i)\$_2$ or $\$_2\text{lab}(e_i)\$_1$ in $G'$. The so-constructed path clearly is a trail from $s$ to $t$ matching $\tilde{r}$ in $G'$ and does not use subpaths labeled $\$_1\sigma\$_1$ or $\$_2\sigma\$_2$. On the other hand, let $p'$ be a trail from $s$ to $t$ that matches $\tilde{r}$ in $G'$. By construction of $G'$ and
the definition of \( \tilde{r} \), \( p' \) is a concatenation of paths of the form \((u, s_1, x)(x, \sigma, y)(y, s_2, v)\) or \((u, s_2, x)(x, \sigma, y)(y, s_1, v)\) for nodes \( u, v \in V \) and \( x, y \in V' \setminus V \) and some symbol \( \sigma \in \Sigma \). By construction of \( G' \), each such path corresponds to a unique edge in \( G \), thus we can replace each such subpath of length 3 with the corresponding edge to obtain a trail from \( s \) to \( t \) matching \( r \) in \( G' \).

Unfortunately, expressions of the form \( \tilde{r} \) are in general neither in \( \text{SP}_{\text{tract}} \) nor in \( T_{\text{tract}} \). Indeed, \( \text{SP}_{\text{tract}}(\tilde{r}) \) and \( T_{\text{tract}}(\tilde{r}) \) are NP-hard in general, but the graph \( G' \) has a very special form. To prove tractability, we first consider a similar expression \( r' \) defined as follows:

\[
r' = h(w_1)((A_1 \cup \{s_1, s_2\})^{2k_1} + \varepsilon)(h(w_2) + \varepsilon)((A_2 \cup \{s_1, s_2\})^{2k_2} + \varepsilon) \cdots \]

\[
(h(w_n) + \varepsilon)((A_n \cup \{s_1, s_2\})^{2k_n} + \varepsilon)h(w_{n+1}).
\]

The connection between \( \tilde{r} \) and \( r' \) is as follows: because of the special form of \( G' \), there is a trail from \( s \) to \( t \) matching \( \tilde{r} \) in \( G' \) if and only if there is a trail \( p' \) from \( s \) to \( t \) matching \( r' \) in \( G' \) and \( p' \) does not have a subpath labeled \( s_1 \sigma s_1 \) or \( s_2 \sigma s_2 \) for any \( \sigma \in \Sigma \).

In order to show that we can decide in polynomial time whether a trail \( p' \) from \( s \) to \( t \) matching \( r' \) that does not contain a subpath labeled \( s_1 \sigma s_1 \) or \( s_2 \sigma s_2 \) for any \( \sigma \in \Sigma \) exists in \( G' \), we adapt the methods used in [MNT20]. Indeed, \( r' \in \text{SP}_{\text{tract}} \) by Theorem 2.5 and since \( \text{SP}_{\text{tract}} \subseteq T_{\text{tract}} \) [MNT20], it follows that \( r' \in T_{\text{tract}} \).

We now adapt some definitions from [MNT20, Section 4.2] to enforce that each shortest trail is "admissible". To this end, let \( N \) be the size of the minimal DFA for \( r' \). We choose \( K = N^2 + 4 \). Let \( \text{Cuts} \) denote the set of non-trivial strongly connected components of the minimal DFA \( A_{r'} = (Q, \Sigma, \delta, Q_I, Q_F) \) for \( r' \). We define an extended abbreviations to be of the form \( \text{Cuts} \times (V \times Q) \times E \). An example is \((C, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1)\). A path \( \pi \) matches \((C, (v, q), e_K e_{K-1} e_{K-2} \cdots e_1)\) if \( \delta_L(q, \pi) \in C \), it starts in \( v \) with prefix \( e_K e_{K-1} \) and its suffix is \( e_{K-2} \cdots e_1 \). We denote this with \( \pi \models (C, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1) \). For an arbitrary set \( E' \) we write \( \pi \models G' (C, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1) \) if \( \pi \models G' (C, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1) \) and all edges of \( \pi \) are from \( E' \cup \{e_1, \ldots, e_K\} \). Let \( p = e_1 \cdots e_m \) be a path and \( r = q_0 \cdots q_m \) be the run of \( A_{r'} \) over \( p \). For a set \( C \) of states of \( A_{r'} \), we denote by \( \text{left}_C \) the first edge \( e_i \) with \( q_i-1 \in C \) and by \( \text{right}_C \) the last edge \( e_j \) with \( q_j \in C \). A strongly connected component \( C \) of \( A_{r'} \) is a long run component of \( p \) if \( \text{left}_C \) and \( \text{right}_C \) are defined and \( |p|_{\text{left}_C, \text{right}_C} > K \). In the extended summary of a trail, every long run component is replaced by an extended abbreviation. An extended candidate summary is an extended summary of the form \( S = e_1 \cdots e_m \) where each \( e_i \) is an edge or an extended abbreviation and all edges occurring in \( S \) are distinct. A path \( p \) that is derived from \( S \) by replacing each \( e_i \) by a trail \( p_i \) such that \( p_i \models e_i \) is called a completion of the (extended) candidate summary \( S \).

Since all paths matching \( s_1 \sigma s_1 \) or \( s_2 \sigma s_2 \) are loops in \( G' \), a shortest path will not use such a subpath. Using this, we can use the NL algorithm from [MNT20, Lemma 4.7] to show the following

**Lemma 3.3.** Let \( r', G' = (V', E', \mathcal{E}') \) be as in the proof of Lemma 3.2(b), \( \alpha = (C, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1) \) be an extended abbreviation, and \( E'' \subset E' \). Then there is an NL algorithm that outputs a shortest trail \( p \) such that \( p \models E'' \alpha \) if it exists and rejects otherwise. Furthermore, if a shortest path \( \pi \) from \( v \) to destination \( e_1 \) with suffix \( e_{K-2} \cdots e_1 \) and with

---

12 We choose this instead of \( K = N^2 \) [MNT20] because we need 4 additional edges to ensure that the path is "long enough" even if we remove some loops.

13 We can adapt the algorithm such that the returned path starts with \( e_K e_{K-1} \).
δ₁(q, π) ∈ C exists, for which π ⊨ E'' α holds and such that π does not contain a subpath labeled $\$₁\sigma\$₁$ or $\$₂\sigma\$₂$, then p does not contain a subpath labeled $\$₁\sigma\$₁$ or $\$₂\sigma\$₂$.

We postpone the proof for readability.

We now turn to local edge domains. Let $E_i$ be as defined in [MNT20, Definition 4.8], that is, $E_1 = E \setminus E(S)$ and $E_{i+1} = E_i \setminus E_{\delta_i}$. We define $E_{\delta_i}$ to be the set of edges used by trails $\pi$ that start with $e_K e_{K-1}$, use only edges in $E_i$, and are of length at most $m_i - K + 2$. A trail $p$ is extended admissible if there exists an extended candidate summary $S = \alpha_1 \cdots \alpha_k$ and trails $p_1, \ldots, p_k$ such that $p = p_1 \cdots p_k$ is a completion of $S$ and $p_i \models_{E_{\delta_i}} \alpha_i$ for every $i \in [k]$. With these notions, we can now show the counterpart of [MNT20, Lemma 4.10], which is the heart of the correctness proof.

Lemma 3.4. Let $r', G'$ be as in the proof of Lemma 3.2(b). Then each shortest trail $p$ from $s$ to $t$ that matches $r'$ in $G'$ and such that no subpath matches $\$₁\sigma\$₁$ or $\$₂\sigma\$₂$ is extended admissible.

We postpone the proof of Lemma 3.4 for readability.

With these ingredients we can now give an NL algorithm similar to [BBG20, Algorithm 1] and [MNT20, MNP21, Lemma 4.12], that is, we enumerate all possible extended candidate summaries $S$ with respect to $(r', G', s, t)$ and apply on each extended summary the following algorithm which consists of four tests:

1. Guess, on-the-fly, a path $p$ from $S$ by replacing each $\alpha_i$ by a trail $p_i$ such that $p_i \models_{E_{\delta_i}} \alpha_i$ for each $i \in [k]$. This test succeeds if and only if this is possible.
2. In parallel, check that $p$ matches $r'$.
3. In parallel, check that $S$ is an extended summary of $p$.
4. In parallel, check that $p$ does not contain a subpath matching $\$₁\sigma\$₁$ or $\$₂\sigma\$₂$ for any $\sigma \in \Sigma$.

If all tests succeed on some candidate summary, then we answer “yes”, and if on each candidate summary at least one test fails, the answer is “no”.

To prove correctness, let there be a shortest trail $p'$ from $s$ to $t$ that does not contain a subpath matching $\$₁\sigma\$₁$ or $\$₂\sigma\$₂$ for any symbol $\sigma$. Then, there is also a shortest such trail, and by Lemma 3.4 this trail is extended admissible. Conversely, if the algorithm succeeds, the path $p$ is a trail because $E(S)$ and the sets $E_{\delta_i}$ are mutually disjoint. By tests (2), (3), and (4), it is a trail from $s$ to $t$ that matches $r'$ and does not contain a subpath matching $\$₁\sigma\$₁$ or $\$₂\sigma\$₂$ for any $\sigma \in \Sigma$.

For the complexity, we note that compared to the NL algorithm in [MNT20, MNP21, Lemma 4.12] we only need to additionally test (4), which can clearly be done in NL.

We now prove (c). On the one hand, $UTrail(a^*ba^*)$ is in polynomial time by Theorem 5.2, while $a^*ba^* \notin T_{tract}$. On the other hand, $(abc)^*$ is in $T_{tract}$ but $UTrail((abc)^*)$ is NP-hard, see Theorem 6.1.

To conclude the proof of part (b), we still need to prove Lemmas 3.3 and 3.4.

Proof of Lemma 3.3. Let $\alpha = (C, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1)$ be an extended abbreviation, and $E'' \subseteq E'$. Let $L = L(r')$. We use the NL algorithm from [MNT20, Lemma 4.7] to obtain a shortest trail $p$ with $p \models_{E''}(C, \text{destination}(e_{K-1}), \delta_L(q, e_{K-1})), e_{K-2} \cdots e_1)$. Then $e_K e_{K-1}p$ is a shortest trail with $e_K e_{K-1}p \models_{E''} \alpha$.

Furthermore, let us assume that there exists a shortest path $\pi$ from $v$ to $\text{destination}(e_1)$ with suffix $e_{K-4} \cdots e_1$ and with $\delta_L(q, \pi) \in C$ exists, $\pi$ does not contain a subpath labeled $\$₁\sigma\$₁$ or $\$₂\sigma\$₂$, and such that $\pi \models_{E''} \alpha$. □
Let $e_K e_{K-1} p = e_K e_{K-1} d_1 \cdots d_n e_{K-2} \cdots e_1$. We assume towards contradiction that $e_K e_{K-1} p$ contains a subpath labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$. Since $\pi$ does not contain such a subpath, it cannot be in $e_{K-2} \cdots e_1$. Thus the subpath(s) labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$ can only be in $e_K e_{K-1} d_1 d_2, d_1 \cdots d_n$, or $d_{n-1} e_{K-2} e_{K-3}$. By definition of $G'$, we can remove the loops labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$ from these to obtain a trail $p''$ from $v$ to destination($e_1$). Since components of $r'$ have the form $A^{\geq k_i}$ for some $k_i \in \mathbb{N}$ and some set of symbols $A$, $\delta_L(q, p'') \in C$. Furthermore, $p''$ has the suffix $e_{K-4} \cdots e_1$. Since $e_K e_{K-1} p$ is a shortest trail with $e_K e_{K-1} p \models e, \alpha$, we have $|e_K e_{K-1} p| = |\pi|$. Since $p''$ is obtained from $e_K e_{K-1} p$ by removing edges, $|p''| < |\pi|$, contradicting the choice of $\pi$. □

Proof of Lemma 3.4. We use the notion of extended abbreviation, extended (candidate) summary, and extended admissible from the proof of Theorem 3.2. The majority of the proof is similar to the proof of [MNT20, MNP21, Lemma 4.10], indeed, we only have to additionally prove that the resulting paths $p'$ do not have subpaths matching $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$ or can be replaced with shorter paths that do not have such subpaths. By $p(e_1, e_2)$ we denote the suffix of $p(e_1, e_2)$ that excludes the first edge (so it can be empty). Notice that $p(e_1, e_2)$ and $p(e_1, e_2)$ are always well-defined for trails.

Let $L$ be the language of $r'$. Let $p = d_1 \cdots d_m$ be a shortest trail from $s$ to $t$ that matches $r'$ in $G'$ and such that no subpath matches $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$. Let $S = \alpha_1 \cdots \alpha_k$ be the extended summary of $p$. Let $p_1, \ldots, p_k$ be trails such that $p = p_1 \cdots p_k$ and $p_i \models \alpha_i$ for all $i \in [k]$. We denote by left$_i$ and right$_i$ the first and last edge in $p_i$. By definition of $p_i$ and the definition of extended summaries, left$_i$ and right$_i$ are identical with left$_C$ and right$_C$ if $\alpha_i \in \text{Abbrev}$ is an extended abbreviation for the component $C$.

Assume that $p$ is not extended admissible. That means there is some edge $e$ used in $p_\ell$ such that $e \notin \text{Edge}_\ell$. There are two possible cases:

1. $e \in \text{Edge}_i$, for some $i < \ell$; and
2. $e \notin \text{Edge}_i$ for any $i$.

In case (1), we choose $i$ minimal such that some edge $e \in \text{Edge}_i$ is used in $p_j$ for some $j > i$. Among all such edges $e \in \text{Edge}_i$, we choose the edge that occurs latest in $p$. This implicitly maximizes $j$ for a fixed $i$. Especially no edge from $\text{Edge}_i$ is used in $p_{j+1} \cdots p_k$.

Let $\alpha_i = (C_i, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1)$. By definition of $\text{Edge}_i$, there is a trail $\pi$ from $v$, starting with $e_K e_{K-1}$ and ending with $e$, with $\delta_L(q, \text{lab}(\pi)) \in C_i$, and that is shorter than the subpath $p[\text{left}_i, \text{right}_i]$ and therefore shorter than $p[\text{left}_i, e]$. Let $\pi$ be a shortest such path.

It was shown in [MNT20, MNP21, Lemma 4.10] that $p' = p_1 \cdots p_{j-1} \pi p(e, d_m)$ is a trail matching $r'$ and that the subpath $e_{K-2} \cdots e_1$ from $(C_j, (v'), q'), e_{K-1} e_{K-2} \cdots e_1)$ is used in $p(e, d_m)$. In order to contradict the choice of $p$, we additionally need that $p'$ does not contain a subpath labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$.

To this end, recall that all paths labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$ in $G'$ are loops.

- By choice of $p$, neither $p_1 \cdots p_{j-1} e_K e_{K-1} e_{K-2} \cdots e_1$ nor $p[e, d_m]$ contain a subpath labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$.
- If $\pi(e_{K-1}, e]$ contained a subpath labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$, then its removal would yield a shorter path $\pi'$ starting with $e_K e_{K-1}$ and ending with $e$, and, by definition of $r'$, with $\delta_L(q, \text{lab}(\pi')) \in C_i$, contradicting the choice of $\pi$.

Thus, only $\pi p(e, d_m)$ could contain subpath(s) labeled $\Sigma_1 \Sigma_2$ or $\Sigma_2 \Sigma_1$: either in the first four edges of $\pi$ or in the last two edges of $\pi$ and the first two of $p(e, d_m)$. For example, $\pi$ could end on $\Sigma_1 a$ while $p(e, d_m)$ starts with $\Sigma_1$. 

THE COMPLEXITY OF RPQS ON UNDIRECTED GRAPHS 17
Proposition 3.6. Let $L$ be a regular language and $F_1, F_2$ be finite languages. Then

(a) if $L \in \text{UT}_{\text{tract}}$, then $F_1 LF_2 \in \text{UT}_{\text{tract}}$ and

(b) if $L \in \text{USP}_{\text{tract}}$, then $F_1 LF_2 \in \text{USP}_{\text{tract}}$.

Figure 3. Expressiveness of USP\textsubscript{tract} and UT\textsubscript{tract}.

Since each path labeled $\$_1\$ or $\$_2\$ is a loop in $G'$, the path obtained from $p'$ by removing all subpaths labeled $\$_1\$ or $\$_2\$ is indeed a path, and more precisely, a trail. By definition of $r'$, subpaths of $p'$ that are labeled $\$_1\$ or $\$_2\$ must be matched by a strongly connected component of $r'$. Thus, in order to prove that $p''$ matches $r'$, we have to prove that enough edges in $C_j$ are used. Thus, it suffices to prove that removing the subpath(s) did not remove an edge of $e'\cdot e'_1$. Since $e'\cdot e'_1$ is in $p(e, d_m)$, and the removal of subpaths matching $\$_1\$ or $\$_2\$ could only remove the first two edges in $p(e, d_m)$, their removal does not affect $e'\cdot e'_1$. Thus $p''$ still matches $r'$.

This concludes case (1). For case (2), we additionally assume without loss of generality that there is no edge $e \in \text{Edge}_I$ that appears in some $p_j$ with $j > i$, that is, no edge satisfies case (1). By definition of $\text{Edge}_I$, there is a trail $\pi$ with $\pi \models e$ that is shorter than $p[\text{left}_I, \text{right}_I]$. We choose $p'$ as the path obtained from $p$ by replacing $p_\ell$ with a shortest such $\pi$.

It was shown in [MNT20, MNP21, Lemma 4.10] that $p' = p_1 \cdots p_\ell \cdot \pi \cdot p_{\ell+1} \cdots p_k$ is a trail matching $r'$. Again, in order to contradict the choice of $p$, we additionally need that $p'$ does not contain a subpath labeled $\$_1\$ or $\$_2\$. Let $(C, (v, q), e_K e_{K-1}, e_{K-2} \cdots e_1) = \alpha_\ell$. Since $p$ does not contain a subpath labeled $\$_1\$ or $\$_2\$, neither $p_1 \cdots p_{\ell-1} e_K e_{K-1}$ nor $e_{K-2} \cdots e_1 p_{\ell+1} \cdots p_k$ contain a subpath labeled $\$_1\$ or $\$_2\$.

Thus, subpaths labeled $\$_1\$ or $\$_2\$ can only occur in $\pi[e_K, e_{K-3}]$. Because all paths labeled $\$_1\$ or $\$_2\$ are loops in $G'$, we can remove these paths from $p$ and the resulting path $p'$ is still a trail. Furthermore, $p'$ is a trail from $v$ that ends with $e_{K-4} \cdots e_1$ and, by definition of $r'$ and its components, with $\delta_L(q, p') \in C$. Since $|e_{K-4} \cdots e_1| = N^2$, [MNT20, MNP21, Lemma 4.3] implies that $\delta_L(q, p') = \delta_L(q, p)$. Thus $p'' = p_1 \cdots p_{\ell-1} \cdot p' \cdot p_{\ell+1} \cdots p_k$ is a trail matching $r'$ that has no subpaths labeled $\$_1\$ or $\$_2\$ and is shorter than $p$, contradicting the choice of $p$.

Since every downward closed language is in $\text{SP}_{\text{tract}}$ by definition, it follows that:

Corollary 3.5. USimPath($L$) and UTail($L$) are solvable in polynomial time for every downward closed language $L$.

We present in Figure 3 an overview of the inclusion properties. The regular expressions provided in this figure can be used to distinguish the classes from one another. For example, the class UT\textsubscript{tract} can be distinguished from the class USP\textsubscript{tract} by the language $a^* bc^*$, while the language $(abc)^*$ is neither in USP\textsubscript{tract} nor in UT\textsubscript{tract}. It is not known if there exists a language in USP\textsubscript{tract} that is not in UT\textsubscript{tract}, which is why we do not give a language in that case.

Proposition 3.6. Let $L$ be a regular language and $F_1, F_2$ be finite languages. Then

(a) if $L \in \text{UT}_{\text{tract}}$, then $F_1 LF_2 \in \text{UT}_{\text{tract}}$ and

(b) if $L \in \text{USP}_{\text{tract}}$, then $F_1 LF_2 \in \text{USP}_{\text{tract}}$.
Proof. To prove (a), let $L \in \text{UTtract}$. Then there exists a P algorithm $A$ which given nodes $x$ and $y$, decides if there is a trail from $x$ to $y$ matching $L$. We can use $A$ to decide if there exists a trail from $s$ to $t$ matching $F_1 LF_2$ as follows: We enumerate over all possible pairs of nodes $(x, y) \in V^2$ and all possible edge-disjoint trails $(p_1, p_2)$ with $p_1$ from $s$ to $x$ matching $F_1$ and $p_2$ from $y$ to $t$ matching $F_2$. Then we use $A$ to decide if there is a trail matching $L$ from $x$ to $y$ in $G$ without the edges in $(p_1, p_2)$.

To prove (b), let $L \in \text{USPtract}$. Then there exists a P algorithm $A$ which given nodes $x$ and $y$, decides if there is a simple path from $x$ to $y$ matching $L$. We can use $A$ to decide if there exists a simple path from $s$ to $t$ matching $F_1 LF_2$ as follows: We enumerate over all possible pairs of nodes $(x, y) \in V^2$ and all possible node-disjoint simple paths $(p_1, p_2)$ with $p_1$ from $s$ to $x$ matching $F_1$ and $p_2$ from $y$ to $t$ matching $F_2$. Then we use $A$ to decide if there is a simple path matching $L$ from $x$ to $y$ in $G$ without the nodes in $(p_1, p_2)$.

As a corollary, all languages definable by simple transitive expressions [MT19], are in $\text{UTtract}$ and $\text{USPtract}$.

4. The Gadget $G_{3\text{SAT}}$ for Lower Bounds

In this section, we construct a gadget for obtaining NP-hardness results throughout the paper. We will reduce from $3\text{SAT}$, which is well known to be NP-complete. An instance is a 3CNF formula $\varphi = \land_{i=1}^{m} C_i$ using variables $\{x_1, \ldots, x_n\}$. The question is if $\varphi$ is satisfiable, that is, there exists an assignment $\alpha : \{x_1, \ldots, x_n\} \rightarrow \{\text{true}, \text{false}\}$ that satisfies $\varphi$. In fact, it is known that $3\text{SAT}$ is NP-complete, even if every variable appears exactly twice negated and twice unnegated in $\varphi$ [DD21].

We will explain how to construct a generic undirected graph $G_{3\text{SAT}}$ that we will later provide with labels to show NP-completeness of $\text{USimPath}(L)$ and $\text{UTrail}(L)$ for various languages $L$. The definition of $G_{3\text{SAT}}$ is somewhat technical\footnote{In fact, some of the reductions in Gourvès et al. [GdLMM12], which use a similar gadget, seem to be flawed, see Appendix A.} and is inspired on a gadget that was used by Eilam-Tzoreff [Eil98] to reduce 3SAT to a variant of the disjoint paths problem with length constraints.

Construction 4.1. (Construction of $G_{3\text{SAT}}$.) Let $\varphi$ be a formula in 3CNF with $m$ clauses and $n$ variables. In the following description, we will sometimes say that we will add a $w$-path from $u$ to $v$ for some word $w$. If $|w| \geq 1$ this means that, between the nodes $u$ and $v$, we will add $|w| - 1$ new nodes and connect them such that the new nodes form a path from $u$ to $v$ that is labeled $w$. If $|w| = 0$, this means that $u$ and $v$ are merged together.

For each clause $(\ell_{i,1} \lor \ell_{i,2} \lor \ell_{i,3})$ in $\varphi$, we construct a clause gadget as in Figure 4 (left). For each variable $x_j$ in $\varphi$, we construct a variable gadget as in Figure 4 (right). The words $w_r$ and $w_o$ are written on the paths in the usual “left-to-right’’ reading direction. We will refer to the edges on $w_r$-paths as red edges.

We now define a switch gadget that we will add for each occurrence of a variable, which leads to $3m$ such gadgets. Let $\ell_{i,k}$ be the $4$th literal in the $i$th clause. We add new nodes $u_{i,k}^1$ and $u_{i,k}^2$ and connect them as follows. We add a $w_b$-path from $u_{i,k}^1$ to $\ell_{i,k}^2$ and from $\ell_{i,k}^1$ to $u_{i,k}^2$. If $\ell_{i,k}$ is the $p$th negated occurrence of variable $x_j$, we additionally add $w_b$-paths from $u_{i,k}^1$ to $x_{j,p}^2$ and from $x_{j,p}^1$ to $u_{i,k}^2$. On the other hand, if $\ell_{i,k}$ is the $p$th unnegated occurrence of variable $x_j$, we additionally add $w_b$-paths from $u_{i,k}^1$ to $\overline{x}_{j,p}^2$ and from $\overline{x}_{j,p}^1$ to $u_{i,k}^2$. \hfill $\square$
Finally, we explain how to connect all gadgets. For each $i \in [m-1]$ we add a $w_o$-path from $c_{i,2}$ to $c_{i+1,1}$, from $c_{m,2}$ to $v_{1,1}$ and for each $j \in [n-1]$ we add a $w_o$-path from $v_{j,2}$ to $v_{j+1,1}$.

We then add $w_o$-paths from $u_{i,1}^2$ to $u_{i,2}^1$, from $u_{j,1}^2$ to $u_{j,2}^1$, and from $u_{j,2}^3$ to $u_{j+1,1}^1$. We set $s_2 = c_{1,1}, t_2 = v_{n,2}, s_1 = u_{1,1}^1$, and $t_1 = u_{m,3}^1$. Finally, we add a $w_m$-path from $t_1$ to $s_2$, new nodes $s$ and $t$, a $w_r$-path from $s$ to $s_1$, and a $w_r$-path from $t_2$ to $t$. We sketch the construction in Figure 5.

\[\text{Figure 4. Clause gadget for the clause } C_i = (\ell_{i,1} \lor \ell_{i,2} \lor \ell_{i,3}) \text{ (left) and variable gadget for } x_j \text{ (right). The paths are labeled such that the words } w_r, w_o \text{ can be read from left to right.}\]

\[\text{Figure 5. Sketch of the extension from two edge-disjoint paths to a single trail matching a language. The concrete placement of the (blue dashed) switch-edges depends on the occurrences of literals in clauses. The arrows indicate the “reading direction” of the words on the paths.}\]

**Theorem 4.1.** Let $w_b, w_r \in \Sigma^+$ and $G_{3\text{SAT}}$ as described in Construction 4.1. The following are equivalent:

(a) $\varphi$ is satisfiable.
(b) There exist node-disjoint paths $p_1$ from $s_1$ to $t_1$ and $p_2$ from $s_2$ to $t_2$ in $G_{3\text{SAT}}$ such that $p_1$ does not use red edges.
(c) If $w_o \neq \varepsilon$ then there exist edge-disjoint paths $p_1$ from $s_1$ to $t_1$ and $p_2$ from $s_2$ to $t_2$ in $G_{3\text{SAT}}$ such that $p_1$ does not use red edges.
(d) There exists a simple path $p$ from $s$ to $t$ in $G_{3\text{SAT}}$ that uses the $w_m$-path from $t_1$ to $s_2$ before using any red edge.
(e) If $w_o \neq \varepsilon$ then there exists a trail $p$ from $s$ to $t$ in $G_{3\text{SAT}}$ which reads the $w_m$-edge before using any red edge.

**Proof.** We first show (b) implies (a). If there exist two node-disjoint paths $p_1$ from $s_1$ to $t_1$ and $p_2$ from $s_2$ to $t_2$ in $G_{3\text{SAT}}$ such that $p_1$ does not use red edges, then $p_1$ has to use all nodes $u_{i,k}^1$ and $u_{i,k}^2$ for each $i \in [m]$ and $k \in [3]$. Since $p_2$ is node-disjoint to $p_1$, it cannot
Figure 6. Example of the reduction from 3SAT with the boolean formula $(x_1 \lor x_1 \lor \overline{x_2}) \land (\overline{x_2} \lor x_3 \lor x_3)$ to the language $da^*c(a + b)^*e$. We use $G_{3SAT}$ with words $w_s = d, w_b = w_o = a, w_m = c, w_r = b, w_t = e$. For readability, we colored the edges of the switch blue (and dashed) and omitted the labels on these edges (which all were $a$). Note that the path starting from $s$ must use the $c$-labeled edge before it can use any of the red edges.

use these vertices, so it can only pass through vertices of clause or variable gadgets. In the variable gadgets, it passes through one of two possible paths, if it passes through $\overline{x_{j,1}}$, we assign the variable $x_j$ the value true, otherwise, we assign it the value false. We prove that this assignment satisfies $\varphi$. If $p_2$ used the path including $\overline{x_{j,1}}$, then $p_1$ cannot use this node, and thus has to use the nodes $\ell_{i,k}^1$ and $\ell_{i,k}^2$ with $\ell_{i,k} = x_j$ instead. Because of this, in clause gadgets, $p_2$ can only use edges which correspond literals which are set to true. Since $p_2$ has to traverse all clause gadgets in order to go from $s_2$ to $t_2$ while avoiding the nodes $u_{i,k}^1$ and $u_{i,k}^2$ for all $i \in [m]$ and $k \in [3]$, it follows that there must be at least one literal with value true in each clause gadget. Thus, $\varphi$ is satisfiable.

The proof that (c) implies (a) is analogous.

We now prove that (a) implies (b) and (c): Let $\theta$ be a satisfying assignment of true values to variables in $\varphi$. We construct a path $p_2$ as follows: in each variable gadget, $p_2$ passes through the path with $\overline{x_{j,1}}$ if $\theta(x_j) = true$ and through $x_{j,1}$ if $\theta(x_j) = false$. In each clause gadget, $p_2$ passes through a path which corresponds to a literal which is set to true. This is possible since there is at least one in each clause. We can then construct a path $p_1$, which is (node- and edge-)disjoint from $p_2$. Between each pair of nodes $u_{i,k}^1$ and $u_{i,k}^2$, there are two possible paths which do not use red edges: one via a clause and one via a variable gadget. Let us assume that $\ell_{i,k}$ represents the literal $\ell$. We have chosen $p_2$ in such a way that it uses at most one of these two edges, so $p_1$ uses the other. If $\ell$ is false, then $p_2$ uses the edge in the variable, but not in the clause gadget. If $\ell$ is true, then $p_2$ does not use the edge in the variable gadget.

Finally, we observe that (b) and (d) are equivalent: From node-disjoint paths $p_1$ and $p_2$ it is straightforward to construct a simple path from $s$ to $t$ by connecting $p_1$ via $w_m$ to $p_2$. On the other hand, we can split a simple path $p$ which does not use red edges before reading $w_m$ into node-disjoint paths $p_1 = p[s_1, t_1]$ and $p_2 = p[s_2, t_2]$.

The proof that (c) and (e) are equivalent is analogous. \qed
Application of $G_{3SAT}$ for 2RPQs. We note that $G_{3SAT}$ can also be used to prove hardness for 2RPQs in which not every symbol is of the form $(a + \bar{a})$. More precisely, we can use $G_{3SAT}$ to show that the 2 disjoint paths problem where one path is directed and one path is undirected is NP-hard, even in a graph without labels/only $a$-labels.

**Lemma 4.2.** Node-/Edge-disjoint $a*/(a + \bar{a})*$-paths in directed graphs is NP-complete.

**Proof.** We use $G_{3SAT}$ with directed paths/edges. More precisely, we direct all $w_b = a$ and $w_o = a$ edges in direction of the usual word, while $w_r = aa$ will be a path of length 2 where the directions point to the node in the middle. Graphically, $w_r$ looks like: $\rightarrow \cdot \rightarrow$.

Then, the directed path from $s_1$ to $t_1$ cannot use $w_r$-paths. The correctness follows similar as in Theorem 4.1.

This implies hardness for several “mixed” 2RPQs like $a*b(a + \bar{a})*$. Interestingly, $G_{3SAT}$ is much less complex than the gadget used in the hardness proof of 2 disjoint paths in the purely directed case [FHW80].

We believe that $G_{3SAT}$ can be used to prove hardness for many more languages, and it would be an interesting direction for future work.

5. Generalizing Two Disjoint Paths

We already discussed the close relationship between 2-disjoint-path problems and UTrail and USimPath in Section 2.6, which we can now make more concrete. Indeed, using $G_{3SAT}$ from Construction 4.1 and Theorem 4.1, we can obtain the following:

**Theorem 5.1.** Let $A$ and $B$ be non-empty subsets of $\Sigma$.

- The node-disjoint $A*/B*$-paths problem on undirected multigraphs is in P if $A = B$, and it is NP-complete otherwise.
- The edge-disjoint $A*/B*$-paths problem on undirected multigraphs is in P if $A = B$ or $A \cap B = \emptyset$, and it is NP-complete otherwise.

**Proof.** For node-disjoint paths: If $A = B$, then we can use the minor theorem (see Proposition 2.8) to find node-disjoint paths in the multigraph restricted to $A$ labels. If $A \neq B$, we can assume without loss of generality that $B \nsubseteq A$ (otherwise rename). Let $a \in A$ and $b \in B \setminus A$. We use $G_{3SAT}$ with $w_b = a$, $w_o = \varepsilon$, $w_r = b$. Since the $A^*$ path cannot use $b$-edges, Theorem 4.1 implies the NP hardness. Since USimPath($L$) is in NP for every regular language, NP-completeness follows.

For edge-disjoint paths: If $A = B$, we can use the minor theorem (see Proposition 2.8) to find edge-disjoint paths in the multigraph restricted to $A$. Otherwise, if $A \cap B = \emptyset$, we can find paths in the subgraph restricted to $A$ or restricted to $B$ separately. If $A \neq B$ and $A \cap B \neq \emptyset$, we can assume without loss of generality that $B \nsubseteq A$ (otherwise rename). Let $a \in A \cap B$ and $b \in B \setminus A$. We use $G_{3SAT}$ with $w_b = a$, $w_o = a$, $w_r = b$. Since the $A^*$ path cannot use $b$-edges, Theorem 4.1 implies the NP hardness. Since UTrail($L$) is in NP for every regular language, NP-completeness follows.

This result can be used to completely classify the complexity of UTrail and USimPath for languages of the form $A^*wB^*$, where $w$ is an arbitrary word. If $w = \varepsilon$, then the language is $A^*B^*$, which is downward-closed and therefore always tractable. The other cases are in the following theorem.
Theorem 5.2. Let \( A, B \) non-empty subsets of \( \Sigma \) and \( w = \sigma_1 \ldots, \sigma_n \in \Sigma^* \) with \( n \geq 1 \). Then \( U\text{SimPath}(A^*wB^*) \) is in \( P \) if:

1. \( A = B \); or
2. \( n = 1 \) and \( A - \{ \sigma_1 \} = B - \{ \sigma_1 \} \); or
3. \( \sigma_1 = \ldots = \sigma_n \) and \( A = \{ \sigma_1 \} \) or \( B = \{ \sigma_1 \} \); or
4. \( \sigma_1 = \ldots = \sigma_i \neq \sigma_{i+1} = \ldots = \sigma_n \) and \( A = \{ \sigma_1 \} \) and \( B = \{ \sigma_n \} \); or
5. \( \sigma_1 = \ldots = \sigma_n \) and \( A = \{ \sigma_1, \sigma_n \} \); or
6. \( \sigma_1 = \ldots = \sigma_{n-1} \neq \sigma_n \) and \( A = \{ \sigma_1 \} \) and \( B = \{ \sigma_1, \sigma_n \} \);

and it is \( NP \)-complete otherwise.

\( U\text{Trail}(A^*wB^*) \) is in \( P \) if one of (1)–(6) holds; or \( A \cap B = \emptyset \); or \( n = 1 \) and \( A \cap B = \{ \sigma_1 \} \); and \( NP \)-complete otherwise.

Proof. We first show that (1)–(6) imply tractability. To this end, we rewrite \( A^*wB^* \) in each case to a language of the form \( w_1C^*w_2C^*w_3 \) or \( w_1C^*D^*w_2 \) for \( C, D \subseteq \Sigma \) and \( w_1, w_2, w_3 \in \Sigma^* \) with \( |w_1| + |w_2| + |w_3| \leq n \). Languages of these forms are tractable because we can enumerate over all possible simple paths/trails matching \( w_1, w_2, w_3 \) and find in the subgraph without \( w_1, w_2, w_3 \) either a path matching a downward closed language, namely \( C^*D^* \), from the end of \( w_1 \) to the start of \( w_2 \), or two node-/edge-disjoint \( C \)-paths from the end of \( w_1 \) to the start of \( w_2 \) and from the end of \( w_2 \) to the start of \( w_3 \) with Theorem 5.1. These tests are in polynomial time since there are at most \( |E|^n \) many edges where \( n \) is a constant, Corollary 3.5, and Theorem 5.1.

In case (1), we can rewrite \( A^*wB^* \) into \( A^*wB^* \), in case (2) into \((A \cup \{ \sigma_1 \})^* \sigma_1 (A \cup \{ \sigma_1 \})^* \), in case (3) into \( \sigma_i^* \sigma_i^* \) or \( A^* \sigma_1^* \), in case (4) into \( \sigma_i^* \sigma_i^* \sigma_{n-i}^* \) for some \( i \in [n] \), in case (5) into \( A^* \sigma_1 A^* \sigma_n^{-1} \), and in case (6) into \( \sigma_1^{n-1}B^* \sigma_n B^* \).

Furthermore, if \( A \cap B = \emptyset \), \( U\text{Trail}(A^*wB^*) \) is also in \( P \) since we can first enumerate over all possible trails matching \( w \) and then find paths matching \( A^* \) and \( B^* \) separately, see Theorem 5.1. The case that \( U\text{Trail}(A \sigma_1 B) \) is in \( P \) if \( A \cap B = \{ \sigma_1 \} \) is more complex and will be proved in Lemma 5.3.

On the other hand, if none of the conditions hold, we can prove \( NP \) completeness:

For every regular language \( L \), \( U\text{Trail}(L) \) and \( \text{SimPath}(L) \) are in \( NP \), thus we only need to prove \( NP \)-hardness. We first prove that if (1)–(6) fail, then \( US\text{imPath}(A^*wB^*) \) is \( NP \)-hard. Since \( A \neq B \) and all rules are symmetric, we can assume without loss of generality that \( B \subseteq A \), that is, \( \exists b \in B \setminus A \). We perform a case distinction on \( w \).

- if \( n = 1 \), then by \( \neg(3) \) we know that there exist \( a \in A, b \in B : a \neq \sigma_1 \neq b \). Since \( B \subseteq A \) and by \( \neg(2) \) \( B \neq A \setminus \{ \sigma_1 \} \), there exist \( a \in A, b \in B \setminus A \) with \( a \neq \sigma_1 \neq b \). We use these symbols to label \( G_{3\text{SAT}} \) as follows: \( w_s = \varepsilon, w_a = a, w_o = \varepsilon, w_{m} = \sigma_1, w_t = b, w_t = \varepsilon \). Since the \( A^* \)-path starting in \( s \) cannot use \( b \)-edges, it has to follow the path until \( t_1 \) (the start of \( \sigma_1 \)). By Theorem 4.1 this implies \( NP \) hardness.
- if \( \exists i < j < k \) with \( \sigma_i \neq \sigma_j \neq \sigma_k \), then take an arbitrary \( a \in A \) and use \( G_{3\text{SAT}} \) with words \( w_s = w_t = \emptyset, w_b = a^n, w_r = b^p, w_o = \varepsilon \) and \( w_m = w \). Since \( w_b \) and \( w_r \) have length \( n \), concatenations of \( w_b \) and \( w_r \) do not yield the substring \( w \).
- if \( \sigma_1 = \ldots = \sigma_n \) and \( A \neq \{ \sigma_1 \} \neq B \): Then there exist \( a \in A \setminus \{ \sigma_1 \} \), and, since \( B \subseteq A \), there is \( b \in B \setminus \{ \sigma_1 \} \) with \( b \neq A \). We then use \( G_{3\text{SAT}} \) with words \( w_s = w_t = \emptyset, w_b = a, w_r = b, w_o = \varepsilon \) and \( w_m = w \).
- if \( \sigma_1 = \ldots = \sigma_i \neq \sigma_j = \ldots = \sigma_n \): due to (4) we distinguish between (a) \( A \neq \{ \sigma_1 \} \) and (b) \( B \neq \{ \sigma_n \} \).
Case (a): If \( A \neq \{\sigma_1\} \), we can pick \( a \in A \setminus \{\sigma_1\} \) and use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = a^n, w_r = b^n, w_o = \varepsilon \) and \( w_m = w \). This works because \( a \neq \sigma_1, b \notin A, \) and \( w \neq b^n \).

Case (b): So let us assume that \( A = \{\sigma_1\} \) and \( B \neq \{\sigma_n\} \). If there exists a \( b \in B \setminus A \) with \( b \neq \sigma_n \), we can use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = \sigma_1, w_r = b, w_o = \varepsilon \) and \( w_m = w \). So let us assume that not such \( b \) exists, that is, \( A = \{\sigma_1\} \) and \( B = \{\sigma_1, \sigma_n\} \). If \( w \) contains 2 \( \sigma_n \), we can use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = \sigma_1, w_r = \sigma_1 \sigma_n \sigma_1, w_o = \varepsilon \) and \( w_m = w \). On the other hand, if \( w \) contains only a single \( \sigma_n \), we are contradicting that (6) fails.

We now prove that if \( A \cap B \neq \emptyset \), and in case \( n = 1 \) additionally \( A \cap B \neq \{\sigma_1\} \), and (1)–(6) fail, then \( UTrail(A^*wB^*) \) is NP-hard. Since \( A \neq B \) and all rules are symmetric, we can assume without loss of generality that \( B \not\subseteq A \), that is, \( \exists b \in B \setminus A \). We perform a case distinction on \( w \).

- if \( n = 1 \), then we additionally know that \( A \cap B \neq \{\sigma_1\} \). Thus, there exists \( a \in A \cap B \) with \( a \neq \sigma_1 \). Together with -(2) and \( B \not\subseteq A \) this implies that there exist \( a \in A \cap B, b \in B \setminus A \) with \( a \neq \sigma_1 \neq b \). We can now prove NP-hardness with \( G_{3SAT} \) using the labels \( w_s = \emptyset, w_b = w_o = a, w_m = \sigma, w_r = b, w_t = \emptyset \). Since \( b \notin A \), and \( a \neq \sigma_1 \neq b \), the \( A \)-path starting from \( s \) is not allowed to use \( b \)-edges before reaching \( t_1 \). Therefore, Theorem 4.1 implies NP hardness.

- if \( \exists i < j < k \) with \( \sigma_i \neq \sigma_j \neq \sigma_k \), then take an arbitrary \( a \in A \cap B \) and use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = a^n, w_r = b^n, w_o = a^n \) and \( w_m = w \). Since \( w_b, w_o, \) and \( w_r \) have length \( n \), concatenations of \( w_b \) and \( w_r \) do not yield the substring \( w \).

- if \( \sigma_1 = \ldots = \sigma_n \) and \( A \neq \{\sigma_1\} \cap B \neq \emptyset \): Then there exist \( a \in A \setminus \{\sigma_1\} \), and, since \( B \not\subseteq A \), there is \( b \in B \setminus \{\sigma_1\} \) with \( b \notin A \). Let \( c \in A \cap B \) arbitrary. We then use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = a, w_r = b, w_o = c \) and \( w_m = w \). This works because \( |w| \geq 2 \) and \( w_o \) never appears twice in a row.

- if \( \sigma_1 = \ldots = \sigma_i \neq \sigma_j = \ldots = \sigma_n \): due to (4) we distinguish between (a) \( A \neq \{\sigma_1\} \) and (b) \( B \neq \{\sigma_n\} \).

Case (a): \( A \neq \{\sigma_1\} \). If \( A \cap B \neq \{\sigma_1\} \), we can pick \( a \in A \cap B \setminus \{\sigma_1\} \) and use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = a^n, w_r = b^n, w_o = a^n \) and \( w_m = w \). This works because \( a \neq \sigma_1, b \notin A, \) and \( w \neq b^n \). If \( A \cap B = \{\sigma_1\} \), then, since \( B \not\subseteq A, B \neq \{\sigma_1\} \). So there exist \( a, b \) with \( \{\sigma_1, a\} \subseteq A \) and \( \{\sigma_1, b\} \subseteq B \). We now perform a case distinction on \( \sigma_n \).

- if \( a = \sigma_n \), we use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = \sigma_1, w_r = b, w_o = \sigma_1 \) and \( w_m = w \). Then \( \sigma_n \) only appears in \( w_m \).

- if \( b = \sigma_n \), we show that \( UTrail(B^*w^{rev}A^*) \) is NP-hard. Therefore, we use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = \sigma_1, w_r = a, w_o = \sigma_1 \) and \( w_m = w^{rev} \). The result for \( UTrail(A^*wB^*) \) then follows since \( UTrail \) is closed under reversal, see Theorem 3.1.

- if \( a \neq \sigma_n \neq b \), we use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = \sigma_1, w_r = b, w_o = \sigma_1 \) and \( w_m = w \). Then \( \sigma_n \) only appears in \( w_m \).

Case (b): So let us assume that \( A = \{\sigma_1\} \) and \( B \neq \{\sigma_n\} \). Since \( A \cap B \neq \emptyset, \sigma_1 \in B \). If there exists a \( b \in B \setminus A \) with \( b \neq \sigma_n \), we can use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = \sigma_1, w_r = b, w_o = \sigma_1 \) and \( w_m = w \). So let us assume that not such \( b \) exists, that is, \( A = \{\sigma_1\} \) and \( B = \{\sigma_1, \sigma_n\} \). If \( w \) contains 2 \( \sigma_n \), we can use \( G_{3SAT} \) with words \( w_s = w_t = \emptyset, w_b = \sigma_1, w_r = \sigma_1 \sigma_n \sigma_1, w_o = \sigma_1 \) and \( w_m = w \). On the other hand, if \( w \) contains only a single \( \sigma_n \), we are contradicting that (6) fails.

Since \( USimPath(L) \) and \( UTrail(L) \) are in NP for every regular language, NP-completeness follows.
To complete the proof of Theorem 5.2 it remains to prove the next lemma.

**Lemma 5.3.** $\text{UTrail}(A^*\sigma B^*)$ is in $P$ if $A \cap B = \{\sigma\}$.

**Proof.** If $A = \{\sigma\}$ or $B = \{\sigma\}$, then tractability follows from condition (3) in Theorem 5.2. So let $a \in A \setminus B$ and $b \in B \setminus A$. We denote by $G_A$ the subgraph of $G$ restricted to edges with labels in $A$ and by $G_B$ the subgraph of $G$ restricted to edges with labels in $B$. We describe a polynomial time algorithm that solves $\text{UTrail}(A^*\sigma B^*)$. Let $G = (V, E, \mathcal{E})$ be an undirected multigraph. We enumerate over all tuples of nodes $(u_1, u_2) \in V \times V$ such that there is at least one $\sigma$-edge between $u_1$ and $u_2$. We name one such edge $e_\sigma$. Let $G'$ be a copy of $G$ without $e_\sigma$ (we delete only a single edge, even if there are multiple $\sigma$-edges between $u_1$ and $u_2$). In $G'$, we rename every $\sigma$-edge

- which is only on a trail from $s$ to $u_1$ path in $G'_A$ and not on a trail from $u_2$ to $t$ path in $G'_B$ to an $a$-edge,
- which is not on a trail from $s$ to $u_1$ in $G'_A$, but on a trail from $u_2$ to $t$ in $G'_B$ to a $b$-edge,
- whose deletion would make $t$ unreachable from $u_2$ in $G'_B$ to a $b$-edge. That is, each $\sigma$-edge separating $u_2$ and $t$ in $G'_B$ is renamed to a $b$-edge.

If, after the renaming, there exist paths (not necessarily disjoint) from $s$ to $u_1$ in $G'_A$ and from $u_2$ to $t$ in $G'_B$, we return true. If, after enumerating over all tuples $(u_1, u_2)$ no such paths were found, we return false.

We now prove correctness. Let us assume the algorithm returned true. Then there exists a tuple of nodes $(u_1, u_2)$, a $\sigma$-edge $e_\sigma$ with endpoints $u_1$ and $u_2$, an $A$-path from $s$ to $u_1$ and a $B$-path from $u_2$ to $t$ in $G'$. Consider an arbitrary trail $p$ from $s$ to $u_1$ in $G'_A$. If $p$ is $\sigma$-free, every path from $u_2$ to $t$ in $G'_B$ will be disjoint from $p$. Thus we can build a trail matching $A^*\sigma B^*$ by concatenating $p$ with $e_\sigma$ and a shortest path from $u_2$ to $t$ in $G'_B$. Otherwise, let $p = e_1 \cdots e_\ell$ and $e_i$ be the first $\sigma$-edge in $p$. Let $x \in \text{Node}(e_\ell)$ be the destination of $e_1 \cdots e_i$. We will construct a trail $p_2$ from $x$ to $t$ in $G'_B$ which does not use $e_i$. Since $e_i$ was the first $\sigma$-edge in $p$ and $A \cap B = \{\sigma\}$, the prefix of $p$ will be disjoint from $p_2$ and therefore $e_1 \cdots e_\ell \cdot p_2$ will be a trail from $s$ to $t$ that matches $A^*\sigma B^*$.

Let $\text{Node}(e_i) = \{x, y\}$. Since $e_i$ was not relabeled, there is a trail from $u_2$ to $t$ in $G'_B$ which uses $e_i$. If this trail can be split into a trail from $u_2$ to $y$ and from $x$ to $t$, then the trail from $x$ to $t$ is our $p_2$. Otherwise, let us assume the $B$-path is split into a trail from $u_2$ to $x$ and one from $y$ to $t$. Since there is a trail from $u_2$ to $t$ not using $e_i$ (otherwise, this edge had been relabeled $b$), we can construct a $B$-path not using $e_i$ by concatenating the path from $x$ to $u_2$ with this path from $u_2$ to $t$. Removing cycles in which edges are used more than once then yields the trail $p_2$ from $x$ to $t$ which does not use $e_i$.

We now turn to the other direction. Let $p = e_1 \cdots e_\ell$ be a trail from $s$ to $t$ matching $A^*\sigma B^*$. Then there exists an $i \in [\ell]$ such that $e_1 \cdots e_{i-1}$ matches $A^*$, $\text{lab}(e_i) = \sigma$, and $e_{i+1} \cdots e_\ell$ matches $B^*$. Since the algorithm enumerates over all tuples of nodes, it will have enumerated over $\text{Node}(e_i)$. Let $(u_1, u_2)$ be the nodes of $e_i$ in the order they appear in $p$. Since all edges in $e_1 \cdots e_{i-1}$ are on a trail from $s$ to $u_1$, they will only be renamed to $a$-edges (if at all). With the same argument, the edges in $e_{i+1} \cdots e_\ell$ will only be renamed to $b$-edges (if at all). Thus, even after renaming the edges in $G'$ as described in the algorithm, there are still paths from $s$ to $u_1$ in $G'_A$ and from $u_2$ to $t$ in $G'_B$. Thus the algorithm will return true.

\[\square\]
6. Word Iterations

In this section we give an overview of the complexity of USimPath and UTrail for word iterations, that is, languages of the form $w^*$, where $w$ is a word. This setting has essentially three cases. The first case, where $w = a^n$ with $n \geq 3$, has been an open problem since 1991 [APY91]. The other two cases are the following.

**Theorem 6.1.** Let $w$ be a word.

(a) If $|w| \leq 2$, then USimPath($w^*$) and UTrail($w^*$) are in P.

(b) If $|w| \geq 3$ and $w$ has at least 2 different symbols, then UTrail($w^*$) and USimPath($w^*$) are NP-complete.

**Proof.** We first prove (a). If $|w| = 1$, finding a simple path or trail is equivalent to finding an arbitrary path. If $|w| = 2$, then we can find simple paths labeled by a word in $L(w^*)$ in P using the graph duplication technique of Edmonds [LP84, Man95]. We note that this technique also works on multigraphs. For trails, if $w = aa$, using the extended line graph construction, the problem reduces to the one for simple paths, see Lemma 2.9. If $w = ab$, then Abouelaoualim et al. [ADF08] give a polynomial time algorithm which builds on the work of Szeider [Sze03] and also works on multigraphs. The case $w = ba$ is equivalent to $w = ab$. We now prove (b). Since USimPath($L$) and UTrail($L$) in NP for every regular language, it remains to prove NP hardness. Assume that $|w| \geq 3$ and $w$ has at least 2 different symbols. We distinguish the following cases:

(1) $w$ is periodic, that is, $w = w_1^i$ for some $i \geq 2$;

(2) $w$ has at least 3 different symbols; or

(3) $w$ is not periodic and has exactly 2 different symbols.

We note that we will use different methods here since we do not see how to handle $w = abab$ with a reduction from 3SAT, while showing hardness for $w = aab$ seems impossible with a reduction from the two node-/edge-disjoint paths problem on directed graphs.

(1) We use a reduction from TwoDisjointPaths or TwoEdgeDisjointPaths, respectively. Both problems are NP-complete on directed graphs [FHW80]. Let the node pairs $s_1, t_1$ and $s_2, t_2$ be given and $G_D = (V_D, E_D, \mathcal{E}_D)$ be a directed graph. Similar to Chou et al. [CMM+94], the main idea is to replace the directed edges with undirected paths labeled with some word which implies the direction. Furthermore, we add paths from a new node $s$ to $s_1$, from $t_1$ to $s_2$, and possibly from $t_2$ to a new node $t$, such that the languages enforce a valid path to take the path from $t_1$ to $s_2$.

For this reduction it is necessary that the paths may not be traversed in the opposite direction. If $w \neq w_{rev}$, we can directly replace each directed edge $e \in E_D$ with an undirected path labeled $w$ from origin($e$) to destination($e$), and add a path labeled $(w_1)^{i-1}$ from $t_1$ to $s_2$ and a new start node $s$ with a path labeled $w_1$ to $s_1$. Furthermore, we define $t = t_2$.

In the case that $w = w_{rev}$, we first need to “shift” $w$. Let $w = w_1^1$. Since $w_1$ has at least two different symbols, we can write $w_1$ in the form $w_tw_r$ such that $w_r$ starts with a symbol which is different from the symbol on which $w_t$ ends. Let $w_2 = w_rw_t$. Then, $w_2 \neq w_2^{rev}$ and $L(w^+ = L(w_r w_t^{\delta_1})^{\delta_2-1} w_r) = L(w_r w_t^{\delta_1})^{\delta_2-1}$. Thus, we replace every edge $e \in E_D$ with an undirected path labeled $w_{\delta}$ from origin($e$) to destination($e$). Furthermore, we add a path labeled $w_t$ from a new node $s$ to $s_1$, an edge labeled $w_{\delta-1}$ from $t_1$ to $s_2$, and an edge labeled $w_r$ from $t_2$ to a new node $t$. Let $G_U$ be the so-constructed undirected graph.

We now show that there are two node-/edge-disjoint paths from $s_1$ to $t_1$ and from $s_2$ to $t_2$ in $G_D$ if and only if there exists a simple path/trail matching $w^*$ from $s$ to $t$ in $G_U$. To
this end, we first observe that for each edge \( e \in E_D \) we find exactly one path of length \( |w| \) from origin(\( e \)) to destination(\( e \)). (Depending on whether \( w = w^{rev} \), this path is either labeled \( w \) or \( w^i \).) We name this path \( \text{corresp}(e) \). We note that for each \( e \in E_D \), \( \text{corresp}(e) \) is simple and does not use any nodes of \( V_D \) besides origin(\( e \)) and destination(\( e \)). Let \( p = e_1 \cdots e_n \) be a path. We define \( \text{corresp}(p) = \text{corresp}(e_1) \cdots \text{corresp}(e_n) \). If there exist two edge-disjoint trails \( p_1 \) from \( s_1 \) to \( t_1 \) and \( p_2 \) from \( s_2 \) to \( t_2 \) in \( G_D \), then we obtain a trail \( p' \) from \( s \) to \( t \) in \( G_U \) matching \( w^* \) by concatenating the path from \( s \) to \( s_1 \), \( \text{corresp}(p_1) \), the path from \( t_1 \) to \( s_2 \), \( \text{corresp}(p_2) \), and the path from \( t_2 \) to \( t \). If \( p_1 \) and \( p_2 \) are node-disjoint simple paths, then the so-constructed path \( p' \) is a simple path by construction. On the other hand, if there is a path \( p' \) from \( s \) to \( t \) in \( G_U \) matching \( w^* \), then it must start with the path from \( s \) to \( s_1 \), and end with the path from \( t_2 \) to \( t \). Since \( p' \) matches \( w^* \), and by definition of \( G_U \), \( p' \) must use the path from \( t_1 \) to \( s_2 \). Furthermore, since \( w \neq w^{rev} \) or \( w_2 \neq w_2^{rev} \), the subpaths \( p_1' \) from \( s_1 \) to \( t_1 \) and \( p_2' \) from \( s_2 \) to \( t_2 \) must follow the paths in the “intended direction”. Let \( p_1 \) and \( p_2 \) be the paths in \( G_D \) obtained from \( p_1' \) and \( p_2' \) by deleting nodes not in \( V_D \) and making its two neighbors adjacent. Furthermore, if \( p' \) is a simple path, then \( p_1 \) and \( p_2 \) must be node-disjoint simple paths. And if \( p' \) is a trail, then \( p_1 \) and \( p_2 \) must be edge-disjoint trails.

(2) Since \( w \) has at least 3 different symbols, say \( a, b, \) and \( c \), we can write it as \( w^+ = w_5(ab'b'c''w')w_6 \) for some words \( w_5, w_6, w_1, \in \Sigma^* \) and numbers \( i, j, k \geq 1 \). We then use \( G_{3SAT} \), see Construction 4.1, with the words \( w_5 = aib', w_6 = b', w_r = w_b = c^kw'a^i, w_1 = b'c^kw', w_m = b' \) to prove NP-hardness.

Note that every path matching \( w^* \) that starts in \( s \) has use the \( w_m \)-path from \( t_1 \) to \( s_2 \) before it can use any red edge because before and after every “shared” \( b'- \)path (that is, every undirected path labeled \( b' \) from \( \ell_{1,k}^j \) to \( \ell_{2,k}^j \), from \( x_{1,k}^j \) to \( x_{2,k}^j \), or from \( y_{1,k}^j \) to \( y_{2,k}^j \)) there is at most one \( a \) and at most one \( c \)-edge and no other \( b \)-edge, see Figure 7 (left). Thus the correctness follows from Theorem 4.1.

(3) Since \( w \) is not periodic and has exactly two different symbols, say \( a \) and \( b \), it can be written as \( w^+ = w_5(ab'b'w')w_6 \) for some words \( w_5, w_6, w_1, \in \{a, b\}^* \) and a number \( i \geq 1 \) such that \( w' \in \varepsilon \) or \( w' \) begins with \( a \) (that is, \( j \) is maximal).

We then use \( G_{3SAT} \) with words \( w_5 = b, w_6 = w_r = b'w'a, w_5 = ab, w_1 = b'b, w_m = b \).

Again, every path matching \( w^* \) that starts in \( s \) has use the \( w_m \)-path from \( t_1 \) to \( s_2 \) before it can use any red edge. The reason can be seen in Figure 7 (right): If the path starting in \( s \) would use a red edge before reading the \( w_m \)-path from \( t_1 \) to \( s_2 \), it would contain a substring \( ab'a \) or \( aba \) instead of \( ab'b \). Thus, this path would not be labeled \( w^* \). Thus the correctness follows from Theorem 4.1. \( \square \)
We consider a simple variant of *chain regular expressions*, which were introduced to study static analysis of schemas for XML [MNS09] and used for studying the complexity of SPARQL query evaluation [LM13].

**Definition 7.1** (Simple Chain Regular Expression (SCRE)). A *factor* is a regular expression of the form $a$, $a^*$, or $a?$ where $a$ is some symbol from $\Sigma$. A *simple chain regular expression (SCRE)* is a (possibly empty) concatenation of factors.

We use a similar shorthand notation for SCREs as in [MNS09]. In short, we write $\text{SCRE}(f_1, \ldots, f_k)$ for the class of SCREs in which we allow factors $f_1, \ldots, f_k$. For example, the expression $a^*b^*ab^*a?$ is in $\text{SCRE}(a, a^?, a^*, b^*)$. We will use a special symbol $\star$ to abbreviate “all alphabet symbols that were not listed yet”. For example, $\text{SCRE}(a, a^*, \star ?)$ is the class of SCREs that use factors in $\{a, a^*\} \cup \{b^* \mid b \in \Sigma - \{a\}\}$. Next, we study $\text{UTrail}(L)$ and $\text{USimPath}(L)$ for languages $L$ that are definable by SCREs.

**Trails are Tractable.** Remarkably, finding trails is tractable for every language definable by an SCRE.

**Theorem 7.2.** $\text{UTrail}(L)$ is in $P$ for every language $L$ definable by an SCRE.

*Proof.* We can write every expression $r \in \text{SCRE}(\star ?, \star, \star)$ in the form $r = r_1 a_1^* r_2 \cdots a_{\ell-1}^* r_{\ell}$, where $r_i \in \text{SCRE}(\star ?, \star)$ for each $i \in [\ell]$. Since $\ell$ is a constant and since each path that matches $r_i$ has constant length, we can iterate in polynomial time over all tuples $(p_1, \ldots, p_\ell)$ of disjoint (sub)trails of the multigraph $G$ such that each $p_i$ matches $r_i$. Let $G'$ be $G$ without the edges of $(p_1, \ldots, p_\ell)$. Assume that path $p_i$ is from $u_i$ to $v_i$ (with $u_1 = s$ and $v_\ell = t$). In order to complete the subtrails to a trail that matches $r$, we will test, for each symbol $a \in \Sigma$ and all $i \in [\ell - 1]$, for edge-disjoint trails in $G'_a$. More precisely, let $k = |\{a_i \mid a_i = a\}|$. For each $a \in \Sigma$, we test if there exist $k$ edge-disjoint paths $p^0_{i_1}, \ldots, p^0_{i_k}$ such that $a_{i_j} = a$ and $p^0_{i_j}$ is a path from $v_{i_j}$ to $u_{i_j+1}$ in $G'_a$. Since $k$ is a constant, their existence can be tested in polynomial time, see Proposition 2.8. Since $G'_{a_i}$ and $G'_{a_j}$ are mutually edge-disjoint graphs for all $a_i \neq a_j$, the so-constructed paths will be edge-disjoint. \hfill \Box

**Simple Paths are Not So Simple.** The situation for finding simple paths is much more complex, however. In order to maintain an overview, we differentiate between the number of alphabet symbols used in the SCREs. Since the number of alphabet symbols in RPQs recently found in query logs is typically low [BMT17, BMT19], even the results on one or two alphabet symbols are of practical interest.

*One or Two Alphabet Symbols.* If the SCREs just use a single alphabet symbol, that is, we have languages definable by an $\text{SCRE}(a, a^?, a^*)$, then $\text{USimPath}$ is always tractable. The next theorem shows that, for a second alphabet symbol $b$, factor types $b, b^?$ or $b^?, b^*$ can be added. We will see later that allowing both $b, b^*$ leads to NP-completeness.

**Theorem 7.3.** $\text{USimPath}(L)$ is in $P$

(a) for every language definable by an $\text{SCRE}(a, a^?, a^*, b, b^?)$ and

(b) for every language definable by an $\text{SCRE}(a, a^?, a^*, b^?, b^*)$. 


Proof. For (a), assume that \( r \in \text{SCRE}(a, a?, a*, b, b?) \). Then there exists an \( \ell \in \mathbb{N} \) with \( r = r_1a_1^*r_2 \cdots a_{\ell-1}^*r_{\ell} \), where \( r_i \in \text{SCRE}(a, a?, b, b?) \) for each \( i \in [\ell] \). Since \( \ell \) is a constant that depends only on \( r \) and since paths that match each such \( r_i \) have constant length, we can iterate in polynomial time over all tuples \((p_1, \ldots, p_\ell)\) of node-disjoint simple (sub)paths such that each \( p_i \) matches \( r_i \). Assume that path \( p_1 \) is from \( u_i \) to \( v_i \) (with \( u_1 = s \) and \( v_\ell = t \)). In order to complete the subpaths to a path that matches \( r \), we need to test if there exist \( \ell - 1 \) paths that are \( a \)-labeled, mutually node-disjoint, node-disjoint from \( p_1, \ldots, p_\ell \), and respectively from \( v_i \) to \( u_{i+1} \), for each \( i \in [\ell - 1] \). Testing if these paths exist can be done by running the polynomial-time algorithm for \( k \)-node-disjoint simple paths on the graph obtained from the input multigraph by deleting all inner nodes of \( p_1, \ldots, p_\ell \). The proof of (b) follows from Lemma 9.5. We now turn to (b). As in (a), we can rewrite every regular expression of this form in a normal form. We then show in Lemma 9.5 that Algorithm 1 correctly decides this problem in polynomial time.

The next few pages are devoted to the proof of Lemma 9.5. We first give some necessary definitions and explain the outline of our proof and the idea of Algorithm 1.

**Observation 8.** Every regular expression \( r \) in \( \text{SCRE}(a, a?, a*, b?, b^*) \) can be written in a normal form

\[
 r = r_1a_1^*b_1^*a_2^*b_2^* \cdots r_{\ell}
\]

with \( a_i \in \{a, \varepsilon\} \), \( b_i \in \{b, \varepsilon\} \), and \( r_i \in \text{SCRE}(a, a?, b?) \). As such, each path that matches \( r \) can be seen as a path that consists of the following subpaths:

- paths \( p_i \) matching \( r_i \) from nodes \( z_{i-1} \) to \( x_i \), for \( i \in [\ell] \),
- \( a \)-paths from nodes \( x_i \) to \( y_i \), for \( i \in [\ell - 1] \), and
- \( b \)-paths from nodes \( y_i \) to \( z_i \), for \( i \in [\ell - 1] \),

where \( z_0 = s \) and \( x_\ell = t \).

Thus USimPath\((L)\) is in \( P \) for every language definable by an \( \text{SCRE}(a, a?, a*, b?, b^*) \) if and only if USimPath\((L)\) can be decided in polynomial time for languages definable by regular expressions of the form \((\dagger)\).

**Definition 8.1.** We say that a multigraph \( H \) is 2-connected if (1) it contains at least two nodes and (2) for each node \( x \), the multigraph \( H - \{x\} \) is connected. A 1-(node-)cut between two nodes \( u \) and \( v \) is a node \( x \) such that every path from \( u \) to \( v \) uses \( x \). A 2-connected component of \( H \) is a subgraph \( C \) of \( H \) that is 2-connected and maximal. Maximal means here that there is no node \( x \notin C \) such that the induced subgraph of \( H \) on \( C \cup \{x\} \) is 2-connected.

We say a path \( p \) hits a 2-connected component \( C \) if an inner node of \( p \) is a node of \( C \).

Furthermore, given a simple path \( p \) and two nodes \( x, y \) in \( p \), we denote by \( p[x, y] \) the subpath of \( p \) from \( x \) to \( y \).

A connection between node-disjoint paths and minimum node cuts was given by Menger [Men27].

**Theorem 8.2** (Menger’s theorem). Let \( u \) and \( v \) be distinct, non-adjacent nodes in a connected, undirected multigraph \( G \). Then the maximum number of internally node-disjoint paths between \( u \) and \( v \) in \( G \) equals the minimum node cut for \( u \) and \( v \), which is the number of nodes, distinct from \( u \) and \( v \), whose removal disconnects \( u \) and \( v \).

\(^{15}\)While Menger worked on graphs, the theorem immediately holds for multigraphs.
A result of Menger’s theorem is that 2-connected components $C$ have two node-disjoint paths between each pair of nodes in $C$.

We now explain how Algorithm 1 can decide USimPath$(r)$ for regular expressions of the form $(\ddagger)$. The outer loop of the algorithm enumerates the tuples $(p_1, \ldots, p_\ell)$ of constant-length node-disjoint simple paths that match the subexpressions of $r_i$, for each $i \in [\ell]$. Once we have these, we can find a simple path that matches $r$ if we can complete $(p_1, \ldots, p_\ell)$ with node-disjoint simple paths that match the subexpressions of the form $a^*$ or $b^*$ at the appropriate places. This problem is essentially the problem of finding disjoint paths where some of the paths need to be labeled $a$ and others need to be labeled $b$. The challenge for the present proof is that this latter problem is NP-complete. Indeed, in a given undirected graph with edge labels $a$ and $b$, deciding if there are two node-disjoint simple paths between $(s_1, t_1)$ and $(s_2, t_2)$, one labeled with $a$’s and the other with $b$’s, is NP-complete [GdLMM12, Theorem 16]. We therefore need a different approach.

Our approach uses a structural graph-theoretic argument to reduce the problem to finding sets of node-disjoint $a$-paths in graphs $G_A^{X,N}$ and sets of node-disjoint $b$-paths in other graphs $G_B^{X,N}$. These graphs will be computed from $G$ based on $p_1, \ldots, p_\ell$ (that is, inner nodes of $p_1, \ldots, p_\ell$ will be removed) and a second much more intricate loop, that we describe later. The crux is that, if these sets of $a$-paths and $b$-paths exist for any $(p_1, \ldots, p_\ell)$ and any $G_A^{X,N}$ and $G_B^{X,N}$, then there is a simple path that matches $r$ in $G$, because $G_A^{X,N}$ and $G_B^{X,N}$ are node-disjoint (up to start/end nodes of paths that we are interested in). If these sets of paths do not exist, then we need to prove a non-trivial re-routing result that shows that no simple path that matches $r$ exists in general. This result (Lemma 8.3) shows that if a simple path that matches $r$ exists, then there exists one that satisfies a number of conditions that allow us to run the inner loop of the algorithm in polynomial time.

We note that in line 4, we call the tuple of nodes $(y_1, \ldots, y_{\ell-1})$ consistent with $(a_i, b_i)_{i\in[\ell-1]}$ if $(x_i = y_i)$ is equivalent to $a_i = \varepsilon$ and $(y_\ell = z_i)$ is equivalent to $b_\ell = \varepsilon$. Thus this line enumerates possible nodes between the $a_i$- and $b_i$-paths.

We now describe the workings of the inner loop.

Let $k$ be the number of occurrences of a factor of the form $a$ in $r$, that is, $k$ is the length of the shortest word in $L(r)$, which is a constant for the purposes of the present decision problem.

Let $G'$ be the multigraph $G$ without the inner nodes of $p_1, \ldots, p_\ell$ and their adjacent edges. For each $i \in [\ell - 1]$ let $G_i$ be the single node $x_i$ if $x_i = y_i$, and the induced subgraph of $G'_a$ on the nodes of simple paths from $x_i$ to $y_i$ otherwise (without the nodes $z_j$ for which $z_j \neq x_{j+1}$ and $y_j \neq z_j$). Notice that in $G_i$ the 1-cuts between $x_i$ and $y_i$ are totally ordered by their proximity to $x_i$. Furthermore, between every pair of such consecutive 1-cut nodes between $x_i$ and $y_i$ in $G_i$, there either is nothing, or

• a 2-connected component with no cycle of length at least $2k$, or
• a 2-connected component with a cycle of length at least $2k$.

We call a 2-connected component of $G_i$ that has a cycle of length at least $2k$ a large component, and otherwise a small component.

We distinguish these components because large components have long simple paths (length at least $k$) between every pair of their nodes, while we can show that small components have simple paths of length at most $4k^2$ (Lemma 9.1).
The crux of our argument is in the following lemma, which says that we can assume that the \(b\)-subpaths of solutions of \(\text{USimPath}(L)\) hit no large component and at most a constant number of small components.

**Lemma 8.3.** Let \(L\) be a language definable by a regular expression of the form (\(\dagger\)). If there exists a solution of \(\text{USimPath}(L)\), then there is a solution \(p_1p_1^b p_2p_2^b \cdots p_t\), such that each \(p_i\) matches \(r_i\), each \(p_i^a\) matches \(a_i^*\), each \(p_i^b\) matches \(b_i^*\) and, furthermore, (1) no \(p_i^b\) hits a large component and (2) all \(p_i\) together hit at most \(\ell(\ell + k)\) different small components.

Our algorithm will therefore consider sets \(X\) that contain at most \(\ell(\ell + k)\) different small components and consider subgraphs \(G_A^{X,N}\) of \(G_a\) through which we will search for disjoint \(a\)-paths. First we construct a set \(A_X\), to which we will add all the nodes \(x_i, y_i,\) all 1-cuts between \(x_i\) and \(y_i\), all nodes of large components between \(x_i\) and \(y_i\), and all nodes of small components \(C \notin X\). For each small component \(C \in X\), observe that, while the size of \(C\) is not necessarily constant,\(^{16}\) a simple path that matches \(r\) traverses \(C\) at most \(\ell - 1\) times. Since simple paths in \(C\) have length at most \(4k^2\), there are at most \(\ell(\ell - 1)4k^2\) nodes per \(C \in X\) that can be used by a simple path that matches \(r\). As such, we can iterate over sets of \(N\) nodes of \(\bigcup_{C \in X} C\), where \(|N| = O(3^k)^3\). For each such subset, we consider the graph \(G_A^{X,N}\) which is the induced subgraph of \(G_a\) by \(A_X \cup N\). The graph \(G_B^{X,N}\) is the induced subgraph of \(G_b\) by all nodes \(y_i, z_i\) (that is, start/end nodes of \(b\)-subpaths) and the nodes that are not in \(G_A^{X,N}\). Our problem can now be solved by finding \(\ell - 1\) node-disjoint paths in \(G_A^{X,N}\) and \(\ell - 1\) node-disjoint paths in \(G_B^{X,N}\).

This concludes the outline of the proof. We will now give useful observations and lemmas. We first observe an important property of languages of the form (\(\dagger\)). Intuitively, these languages are special because we can replace substrings with a “long enough” sequence of the symbol \(a\). Let \(k\) be the length of the shortest word in \(L(r)\).

**Observation 9.** Let \(r\) be a regular expression of the form (\(\dagger\)). Then each word \(w \in L(r)\) can be written in the form \(w_1w_1^a w_2^b \cdots w_{\ell-1} w_{\ell-1}^a w_\ell^b\) such that \(w_i\) matches \(r_i\), and \(w_i^a\) matches \(a_i^*\), and \(w_i^b\) matches \(b_i^*\). Let \(x, y \in \{a, b\}\). If, for some \(i, j \in \{1, \ldots, m\}\), \(x_i = x\) and \(y_j = y\), then the word obtained by replacing the substring between \(w_i^x\) and \(w_j^y\) (and arbitrary parts of \(w_i^x\) and \(w_j^y\)) with any word in \(a^{\geq k_{ij}}\) where \(k_{i,j}\) is the number of factors \(a\) between \(x_i^*\) and \(y_j^*\) in \(r\), is in \(L(r)\). Let \(k\) be the number of factors \(a\) in \(r\). Since \(k_{i,j} \leq k\) for every \(i, j\), any word in \(a^{\geq k}\) can be used.

We now prove that the length of simple paths in 2-connected components without a cycle of length at least \(2k\) is bounded.

**Lemma 9.1.** For each 2-connected component \(C\) without a cycle of length at least \(2k\) it holds that the length of the longest simple path between all pairs \(v_1, v_2 \in C\) is at most \((2k)^2\).

**Proof.** Let us assume towards contradiction that there exist two nodes \(s, t\) with a simple path \(p\) of length at least \((2k)^2 + 1\) between them. Due to the 2-connectedness and Menger’s theorem, see Theorem 8.2, there exist two node-disjoint paths \(p_1, p_2\) from \(s\) to \(t\). If \(|p_1| + |p_2| \geq 2k\), we find a cycle of length at least \(2k\) by concatenating \(p_1\) and \(p_2\). Thus we can assume without loss of generality that \(|p_1| + |p_2| < 2k\). This implies that \(|p_1| < 2k\). We now prove that there exist nodes \(x, y\) in \(p_1\) and \(p\) such that (1) \(|p[x, y]| \geq 2k\) and (2) \(p_1[x, y]\) and \(p[x, y]\) are node-disjoint (up to \(x\) and \(y\)).

\(^{16}\)Take \(C\) with edges \((s, i), (i, t)\) with \(i \in [n]\) for arbitrarily large \(n\).
ALGORITHM 1: Deciding USimPath(L) for L of the form ($\dagger$)

Input: Undirected Multigraph $G = (V, E, \mathcal{E})$, nodes $z_0, x_\ell$
Output: “Yes” if there is a simple path from $z_0$ to $x_\ell$ in $G$ that matches $L$; “no” otherwise

1. $k \leftarrow$ length of the shortest word in $L$  

2. foreach tuple of simple paths $\bar{p} = (p_1, \ldots, p_\ell)$ matching $(r_1, \ldots, r_\ell)$ do  
3. $G' \leftarrow G$ without inner nodes of $(p_1, p_2, \ldots, p_\ell)$ and their adjacent edges  
4. foreach tuple $(y_1, \ldots, y_{\ell-1})$ consistent with $(a_i, b_i)_{i \in [\ell-1]}$ do  
5. $A \leftarrow \emptyset$  
6. $S \leftarrow \emptyset$  
7. foreach $i \in [\ell - 1]$ do  
8. foreach node $v$ in a 1-cut between $x_i$ and $y_i$ in $G_i$ do  
9. add $v$ to $A$  
10. foreach large component $C$ between $x_i$ and $y_i$ in $G_i$ do  
11. add each node of $C$ to $A$  
12. foreach small component $C$ between $x_i$ and $y_i$ in $G_i$ do  
13. add $C$ to $S$  
14. foreach small component $C \in S$ do  
15. $I_C \leftarrow \{ i \mid C$ is a small component between $x_i$ and $y_i$ in $G_i \}$  
16. foreach $X \subseteq S$ with $|X| \leq \ell \cdot (\ell + k)$ do  
17. $A_X \leftarrow \emptyset$  
18. foreach $C \in S - X$ do  
19. add each node of $C$ to $A_X$  
20. $N \leftarrow \emptyset$  
21. foreach set of $|I_C|$ disjoint simple paths between $(s_i^C, t_i^C)_{i \in I_C}$ with $C \in X$ do  
22. add each node of these paths to $N$  
23. $G_A^X.N \leftarrow$ induced subgraph of $G_a$ on $(A \cup A_X \cup N) \cup \{(x_i, y_i)_{i \in [\ell-1]}\}$  
24. $G_B^X.N \leftarrow$ induced subgraph of $G_b$ on $(V' - (A \cup A_X \cup N)) \cup \{(y_i, z_i)_{i \in [\ell-1]}\}$.  
25. if there are node-disjoint simple paths between $(x_i, y_i)_{i \in [\ell-1]}$ in $G_A^X.N$ and there are node-disjoint simple paths from between $(y_i, z_i)_{i \in [\ell-1]}$ in $G_B^X.N$ then  
26. return “yes”  
27. return “no”

The proof is due to the lengths of $p$ and $p_1$. Since $p_1$ and $p$ are paths from $s$ to $t$, they are not node-disjoint. And as $p_1$ has at most $2k$ nodes while $p$ has length at least $(2k)^2 + 1$, there must be a subpath $p'$ of $p$ of length at least $2k$ which is node-disjoint from $p_1$ (up to its endpoints). If we choose a maximal such subpath, we can choose its endpoints as $x$ and $y$.

Thus we obtain a cycle of length at least $2k$ by joining $p[x, y]$ and $p_1[x, y]$, which leads to a contradiction. \hfill $\square$
From this we easily obtain the following lemma, which we will need to bound the running time of our algorithm.

**Lemma 9.2.** In a 2-connected component \( C \) with \( n \) nodes and without a cycle of length at least \( 2k \) there are at most \( n^{(2k)^2} \) many different simple paths between every pair of nodes \( v_1, v_2 \in C \).

**Proof.** As \( C \) has no cycle of length at least \( 2k \) and is 2-connected, the length of the longest path in \( C \) is at most \( (2k)^2 \), see Lemma 9.1. As in each step there are at most \( n \) choices for the next node, this yields at most \( n^{(2k)^2} \) many different simple paths from \( v_1 \) to \( v_2 \).

On the other hand, we can show that 2-connected components with a cycle of length at least \( 2k \) will always have a path of length at least \( k \) between each node-pair.

**Lemma 9.3.** In a 2-connected multigraph \( G \) with a cycle of length at least \( 2k \) there is a simple paths of length at least \( k \) between each pair of nodes \( v_1, v_2 \in G \).

**Proof.** Let \( C \) be a cycle of length at least \( 2k \). Let \( v_1, v_2 \) be two nodes in \( G \). We perform a case distinction on the relation of \( v_1 \) and \( v_2 \) regarding \( C \). If they both are on \( C \), we clearly have a simple path of length at least \( k \) between them—we can always choose the longer arc of \( C \).

If both are not on \( C \), we can choose two nodes \( y_1 \) and \( y_2 \) from \( C \). Due to the 2-connectedness we find a simple path \( p_1 \) from \( v_1 \) to \( y_1 \). We can assume without loss of generality that \( y_1 \) is the first hit of this simple path with \( C \) (otherwise rechoose \( y_1 \)). Due to the 2-connectedness, there are two node-disjoint paths from \( v_1 \) to \( y_2 \)—so at least one of them avoids \( y_1 \). We name one of the paths that avoids \( y_1 \) \( p_2 \). Again, let \( y_2 \) be the first hit of \( p_2 \) with \( C \) (otherwise rechoose \( y_2 \)). Also, there is a simple path \( p \) from \( v_2 \) to \( y_2 \). If this one does not hit \( p_1 \) or \( p_2 \), we simply choose this path (and rechoose \( y_2 \) so that it is the first node of \( p \) which is in \( C \)) if \( p \) hits \( p_1 \) or \( p_2 \), we can re-route, using whichever is hit first. Clearly, we obtain two node-disjoint paths from \( \{v_1, v_2\} \) to \( \{y_1, y_2\} \) which are node-disjoint from \( C \), therefore, we can again route via the longer arc of \( C \).

It remains to consider the case where one of \( \{v_1, v_2\} \) is on \( C \) and the other is not. Let without loss of generality \( v_1 \in C \). Due to the 2-connectedness there exists a node \( y \neq v_1, y \in C \) and two node-disjoint paths from \( v_2 \) to \( y \). Since they are disjoint, only one can use \( v_1 \), so we route over the other and possibly rechoose \( y \) to the first hit with \( C \). Then we can use the path from \( v_2 \) to \( y \) and from \( y \) the long arc of \( C \) to \( v_1 \) to construct a path of length at least \( k \).

Before we continue, we need some more notation. We already defined 1-cuts and 2-connected components in Definition 8.1. For clarification and readability, we repeat some definitions given in the outline and add a few new ones. Let \( z_0, x_1, y_1, z_1, \ldots, x_{\ell-1}, y_{\ell-1}, z_\ell \) be distinguished nodes in \( G \). Let \( (p_1, \ldots, p_\ell) \) be a tuple of constant-length node-disjoint simple paths from \( z_{i-1} \) to \( x_i \) that match the subexpressions of \( r_i \), for each \( i \in [\ell] \). Let \( G' \) be the induced multigraph obtained from \( G \) after removing the inner nodes of \( (p_1, \ldots, p_\ell) \).

Let \( G'' \) be the induced multigraph obtained from \( G' \) after additionally removing the nodes \( z_i \) unless \( z_i = x_{i+1} \) or \( z_i = y_i \). (That is, we remove the start/end-nodes which do not belong to \( a_i \)-paths.)

We define \( G_i \) to be the induced subgraph of \( G'' \) which contains \( x_i \) and \( y_i \) and, if \( x_i \neq y_i \), all nodes which are on simple paths from \( x_i \) to \( y_i \). If \( x_i = y_i \), then \( G_i \) contains only the node \( x_i \) and no edges. We observe that \( G_i \) depends only on \( G \), non-empty paths \( (p_j)_{j \in [\ell]}, x_i \) and
exists a solution of \( p \) large component and (2) all \( x \) which is allowed by Observation 9. We will find long \( a \) uses at least one \( \pi \) a \( b \) solution satisfying (1). Let \( C \)\( s \) we name the leftmost node of a 2-connected component \( t \), the graph \( G \) resembles a “string-of-beads”, or a single bead if there is a 2-connected component containing \( x \) and \( y \).

We name a 2-connected component \( C \) in \( G \) large component if it contains a cycle of length at least \( 2k \) and small component otherwise. On this “string of beads” from \( x \) to \( y \), we name the leftmost node of a 2-connected component \( C \)\( s \), and the rightmost one \( t \). All \( s \) and \( t \) are 1-cuts separating \( x \) and \( y \) (or \( x \), \( y \) themselves) by construction. Note that a 2-connected component \( C \) can be between different nodes \( x \), \( y \) and \( x, y \), therefore, there can be different nodes \( s, t \) and \( s^j, t^j \) for \( i \neq j \).

We now have the ingredients to prove Lemma 8.3 which is restated here for readability:

**Lemma 9.4.** Let \( L \) be a language definable by a regular expression of the form (1). If there exists a solution of \( USimPath(L) \), then there is a solution \( p_1p_2p_3p_4\cdots \) such that each \( p_i \) matches \( r_i \), each \( p_i^a \) matches \( a_i \), each \( p_i^b \) matches \( b_i \) and, furthermore, (1) no \( p_i \) hits a large component and (2) all \( p_i \) together hit at most \( \ell(k+1) \) different small components.

**Proof Sketch.** We start with a solution \( p \) where the length of the \( p_i \)'s is as short as possible. If \( p \) contradicts (1) or (2), we successively replace contradicting subpaths with long \( a \)-paths, which is allowed by Observation 9. We will find long \( a \)-paths in large components with Lemma 9.3 or, if many small components in a row are hit, by using a long \( a \)-path via the small components in the middle. Examples of reroutings are depicted in Figure 8. When replacing subpaths, the newly generated path \( \pi \) will have at least one \( b \)-edge less than \( p \), thus contradicting the choice of \( p \).

**Proof.** Let \( a_1^1b_1^1a_2^2b_2^2\cdots a_i^ib_i^i \) be given and let \( p \) be a solution such that the length of the \( p_i \)'s is as short as possible. Let \( x_1, \ldots, x_i, y_i, \ldots, y_t, z_0, \ldots, z_t \) be the respective nodes in \( p \), such that each path from \( x_i \) to \( y_i \) is a \( a \)-path, each path from \( y_i \) to \( z_i \) is a \( b \)-path, and each path from \( z_{i-1} \) to \( x_i \) matches \( r_i \).

We will show that restriction (1) is valid, that is, if there is solution \( p \), then there is a solution satisfying (1). Let \( C \) be a large component between \( x_h \) and \( y_h \), that is hit by some \( b \)-path. Let \( x \) be the first node in \( p \) that is also in \( C \) and \( y \) the last one. Lemma 9.3 ensures that there is an \( a \)-path of length at least \( k \) between all nodes in a large component. Let \( \pi[x,y] \) be an \( a \)-path of length at least \( k \) from \( x \) to \( y \) in this large component. Then \( \pi[x,y] \) uses at least one \( b \)-edge less than \( p[x,y] \) since it avoids at least some part (at least the first or last edge) of the \( b \)-path that hits \( C \). Let \( \pi = p[s,x] \cdot \pi[x,y] \cdot p[y,\ell] \). The so constructed path \( \pi \) is a simple path by construction and matches \( L \), see Observation 9. Furthermore, \( \pi \) contains less \( b \)-edges than \( p \), contradicting the choice of \( p \).

Let us now assume that (1) holds. We show that there is a solution for which (1) and (2) hold. So let us assume that all \( p_i^h \) together hit at least \( \ell(\ell+k)+1 \) different small components. Then there exist \( x_h, y_h \) such that at least \( \ell+k+1 \) small components between them are hit. By definition of \( p \), \( x_h \) and \( y_h \), in every small component \( C \) from \( x_h \) to \( y_h \) there is a path \( \pi^h \) from \( s^h \) to \( t^h \), which only uses nodes in \( p_i^h \). We now observe that each \( p_i^h \) with \( i \neq h \) can hit at most one small component from \( x_h \) to \( y_h \) by definition of 2-connected component. Thus there are at least \( k+2 \) small components
that are hit by some $p^b_i$ path, but not by any $p^b_j$ with $h \neq j$. We name the second such component, $C_x$, and an other such component which is at least $k - 1$ small components closer to $y_h$, $C_y$. We note that this implies that $p[t_{C_x}^{h_i}, s_{C_y}^{h_j}]$ is a subpath of $p^b_h$ of length at least $k$.

Since $C_x$ is only used by $b$-paths and $p^b_i$, in $C_x$ there must exist either
(a) a node $x$ in $p[s, x_h]$ and in $C$ which is part of a $b$-path and such that there exists an $a$-path in $C_x$ from $x$ to $t_{C_x}^{h_i}$ not using any inner nodes in $p^b_i$ for any $i$, or
(b) a node $x'$ in $p[y_h, t]$ and in $C$ which is part of a $b$-path and such that there exists an $a$-path in $C_x$ from $x'$ to $t_{C_x}^{h_i}$ not using any inner nodes in $p^b_i$ for any $i$.

Analogously for $C_y$, there must exist a node $y$ in $p[s, x_h]$ or a node $y'$ in $p[y_h, t]$ which is part of a $b$-path and such that there exists an $a$-path from $y$ to $s_{C_y}^{h_j}$ in $C_y$ (or from $y'$ to $s_{C_y}^{h_j}$, respectively) not using any inner nodes in $p^b_i$ for any $i$.

We perform a case-distinction, depending on which of the nodes $x, x', y, y'$ exist. By definition of $C_x$, either $x$ or $x'$ has to exist and by definition of $C_y$, either $y$ or $y'$ has to exist.

- If $x$ exists, we can reroute as in Figure 8(top), that is, we obtain the simple path

$$\pi = p[s, x] \cdot \pi_1 \cdot p[t_{C_x}^{h_i}, t],$$

where $\pi_1$ is the $a$-path from $y$ to $t_{C_x}^{b_h}$ which does exist by definition of $y$ and is therefore node-disjoint from $p[s, x]$ and $p[t_{C_x}^{h_i}, t]$ (up to $y$ to $t_{C_x}^{h_i}$).

- If $y'$ exists, we can reroute as in Figure 8(mid), that is, we obtain the simple path

$$\pi = p[s, s_{C_y}^{h_j}] \cdot \pi_1 \cdot p[y', t],$$

where $\pi_1$ is the $a$-path from to $s_{C_y}^{h_j}$ to $y'$ which exists by definition of $y'$ therefore node-disjoint from $p[s, s_{C_y}^{h_j}]$ and $p[y', t]$ (up to $s_{C_y}^{h_j}$ and $y'$).

- If $x'$ and $y$ exist, we can rerout as in Figure 8(bottom), that is, we obtain the simple path

$$\pi = p[s, y] \cdot \pi_1 \cdot (p[t_{C_x}^{h_i}, s_{C_y}^{h_j}])^{rev} \cdot \pi_2 \cdot p[x', t],$$

where $\pi_1$ and $\pi_2$ exist by definition of $x'$ and $y$ and are node-disjoint from each other and from $p[s, y], p[t_{C_x}^{h_i}, s_{C_y}^{h_j}]$, and $p[x', t]$ (up to start/end-nodes). We note that $(p[t_{C_x}^{h_i}, s_{C_y}^{h_j}])^{rev}$ is the subpath $p[t_{C_x}^{h_i}, s_{C_y}^{h_j}]$ read from right to left.

We note that in all cases, Observation 9 ensures that $\pi$ matches $L$. Furthermore, $x, x', y, y'$ are inner nodes of $b$-paths. Thus, in each rerouting, we omitted at least the first or last $b$-edge of at least one of these $b$-paths (namely one of the $b$-paths through $x, x', y, y'$). This implies that each $\pi$ is a solution and has at least one $b$-edge less than $p$, contradicting the choice of $p$.

With this we can finally prove the following lemma, which implies Theorem 7.3(b).

**Lemma 9.5.** Algorithm 1 works correctly and in polynomial time for fixed languages.

**Proof.** We first explain why Algorithm 1 is in P:
- The paths $(p_1, \ldots, p_\ell)$ have constant length and thus all possibilities can be enumerated in line 2 in polynomial time.
- One can enumerate over all possible tuples $(y_1, \ldots, y_{\ell-1})$ consistent with $(a_i, b_i)_{i \in [\ell-1]}$ in line 4 in polynomial time. Indeed, we can enumerate all possible tuples with $x_i = y_i$ if $a_i = \varepsilon$ and with $y_i = z_i$ if $b_i = \varepsilon$ (which is exactly the definition of consistency).
Figure 8. Three example reroutings in Lemma 8.3. The dashed circles depict small components, and the red edges exist by definition of $x, x', y,$ or $y'$. The new paths follow the thick edges from $s$ to $t$.

- Given $(p_1, \ldots, p_\ell), x_i, y_i$, one can construct $G_i$ in polynomial time: If $x_i = y_i$, it is only a single node. Otherwise, if $x_i \neq y_i$, we start with $G_\alpha$ and remove some nodes depending on $(p_1, \ldots, p_\ell)$. More precisely, we remove all inner nodes of $(p_1, \ldots, p_\ell)$ and all end-nodes if they do not coincide with $(x_i, y_i)$. We can then determine all nodes on simple paths from $x_i$ to $y_i$ by adding an edge from $x_i$ to $y_i$ (if it does not already exist) and using the polynomial time algorithm of Hopcroft and Tarjan [HT73] to determine (all) biconnected components in this multigraph.
- The test whether $v$ is in a 1-cut between $x_i$ and $y_i$ in line 8 can be done by testing reachability from $s$ to $t$ in $G_i$ and in $G_i - \{v\}$.
- To test in line 10 if $C$ is a large component, it suffices to test if $C$ has a cycle of length $2k$ as a minor.
- $\ell \cdot (\ell + k)$ is a constant, and one can enumerate over all possible subsets of constant size in polynomial time in line 16.
- For each small component $C$, there are only polynomially many simple paths from $s^i_C$ to $t^i_C$ for each $i$, see Lemma 9.2. Since $|I_C| \leq \ell$, we can enumerate over all polynomially many choices of paths in line 21.
- One can test for $\ell$ node-disjoint paths in line 25 in polynomial time, see Proposition 2.8.

We now prove correctness: If Algorithm 1 answers “yes”, we can construct a solution in an obvious way.

For the other direction, let us assume there is a solution to USimPath($L$). Then Lemma 8.3 guarantees that there exists a solution $p = p_1 p_2^a p_2^b p_3^a p_3^b \cdots p_\ell$ such that $p_i$
matches \(r_i\), each \(p^a_i\) matches \(a_i^*\), each \(p^b_i\) matches \(b_i^*\) and, furthermore, (1) no \(p^b_i\) hits a large component and (2) all \(p^b_i\) together hit at most \(\ell(\ell + k)\) different small components, where \(k\) is the length of the shortest path in \(L\).

Since the algorithm checks for each possible combination of paths \((p_1, \ldots, p_\ell)\) and nodes \((y_i)_{i \in [\ell - 1]}\) if a solution of this form exists, it will return the correct result.

This concludes the proof of Theorem 7.3(b).

A natural question is now if the algorithms in Theorem 7.3 can be combined to show that \(\text{USimPath}(L)\) is in P for every language definable by \(\text{SCRE}(a, a^*, b, b^*, b^*)\). This, however, is not the case, even for \(\text{SCRE}(a, a^*, b, b^*)\). The following can be obtained by using \(G_{3\text{SAT}}\) with \(w_s = \varepsilon, w_b = b, w_o = \varepsilon, w_r = aa, w_m = ab, w_t = \varepsilon\).

**Proposition 9.6.** \(\text{USimPath}(b^*abb^*a^*)\) is NP-complete.

**Proof.** \(\text{USimPath}(b^*abb^*a^*)\) is trivially in NP. To prove NP hardness, we use \(G_{3\text{SAT}}\) from Construction 4.1 with \(w_s = \varepsilon, w_b = b, w_o = \varepsilon, w_r = aa, w_m = ab, w_t = \varepsilon\). Since the “red edges” are labeled \(aa\) and the \(w_m\)-path is the only occurrence of a single \(a\) followed by \(b\), every simple path from \(s\) to \(t\) which matches \(b^*abb^*a^*\) has to read the \(w_m\)-path before reading a red edge. Thus NP hardness follows from Theorem 4.1.

On the other hand, it is also not the case that every language that uses all the factors \(a, a^*, b, b^*\) is NP-hard. An obvious example is \(a^+b^+\), and a more intriguing one is summarized in the following theorem.

**Theorem 9.7.** \(\text{USimPath}(a^+b^+a^+b^*)\) is in P.

**Proof Sketch.** We reduce the problem to two calls to the 2 node-disjoint paths problem. We iterate over all triples \((x_1, x_2, x_3)\) of nodes. For each such triple, we compute two sets of nodes \(A\) and \(B\). The former set should be avoided by \(b\)-paths and the latter by \(a\)-paths. We add each 1-cut node between \(s\) and \(x_1\) in \(G_a\) to \(A\) and each 1-cut node between \(x_3\) and \(t\) in \(G_b\) to \(B\). Intuitively, if a path \(p\) labeled \(a^+b^+a^+b^*\) is supposed to have label alternations at \(x_1\), \(x_2\), and \(x_3\), then the nodes in \(A\) (respectively, \(B\)) need to be used by \(a\)-labeled (respectively, \(b\)-labeled) subpaths of \(p\). If \(A\) and \(B\) intersect, we move to the next triple \((x_1, x_2, x_3)\). The algorithm tests if, for some triple, \(G_a - B\) has two node-disjoint paths between \((s, x_1)\) and \((x_2, x_3)\) and \(G_b - A\) has two node-disjoint paths between \((x_1, x_2)\) and \((x_3, t)\). Correctness is non-trivial.

**Proof.** We prove in Lemma 10.5 that this problem can be solved with the algorithm in the proof sketch, which is also depicted as Algorithm 2.

In order to prove Theorem 9.7, we show that Algorithm 2 solves \(\text{USimPath}(a^+b^+a^+b^*)\) and is in P. Since the algorithm starts with enumerating nodes \(x_1, x_2, x_3\), we assume in the following lemma that \(G\) is an undirected multigraph, and \(x_1, x_2, x_3\) are nodes in \(G\).

We start with some notation. Let 1-cut and 2-connected component be as defined in Definition 8.1. By 2-connected components of \(G_a\) from \(s\) to \(x_1\) we refer to the 2-connected components in the subgraph of \(G_a\) induced by the nodes of simple paths from \(s\) to \(x_1\), and by 2-connected components of \(G_b\) from \(x_3\) to \(t\), we refer to the 2-connected components in the subgraph of \(G_b\) induced by the nodes of simple paths from \(x_3\) to \(t\). (Note that a 2-connected component of \(G_a\) could contain \(t\).) If \(s = x_1\) there are no 2-connected components in \(G_a\) from \(s\) to \(x_1\), and if \(x_3 = t\) there are no 2-connected components in \(G_b\) from \(x_3\) to \(t\). Let \(A\)
be the set of all nodes which are 1-cuts between $s$ and $x_1$ in $G_a$ and let $B$ be a set disjoint from $A$ which stores all 1-cuts between $x_3$ and $t$ in $G_b$.

We say that a path $p$ touches a 2-connected component $C$ if a node of $p$ is in $C$ (note that this can also refer to the start or end-node of $p$).

Given $x_1, x_2, x_3$, by $p_{a1}$ we will always denote an $a$-path from $s$ to $x_1$, by $p_{b1}$ a $b$-path from $x_1$ to $x_2$, by $p_{a2}$ an $a$-path from $x_2$ to $x_3$, and by $p_{b2}$ a $b$-path from $x_3$ to $t$.

The following observation is very important for our algorithm:

Observation 10. If there exist $x_1, x_2, x_3$ such that there exist 2-disjoint $a$-paths from $s$ to $x_1$ and from $x_2$ to $x_3$, and there exist 2-disjoint $b$-paths from $x_1$ to $x_2$ and from $x_3$ to $t$, and the paths $s$ to $x_1$ and from $x_3$ to $t$ do not intersect, then there is a simple path matching $a^*b^+a^+b^*$ from $s$ to $t$.

Therefore, we will focus mostly on the paths $p_{a1}$ and $p_{b2}$.

Lemma 10.1. Let $C_1$ be 2-connected component of $G_a$ from $s$ to $x_1$ and $p$ be a simple $b$-path ending in $t$ in $G_b - A$ which

(1) touches $C_1$ in at least 3 nodes, or

(2) touches $C_1$ and another 2-connected component $C_2$ of $G_a$ from $s$ to $x_1$ both at least twice.

Then there is a simple path matching $a^*b^+a^+b^*$ from $s$ to $t$.

Proof. Let the $b$-path $p$ be fixed. Let $C_1$ be the first 2-connected component of $G_a$ from $s$ to $x_1$ that is touched at least twice. That is, we choose $C_1$ as close to $s$ as possible. We perform a case distinction on whether $C_1$ is touched twice or more often. Case 1: $C_1$ is touched at least three times. If there are more touch points, we choose $v_1, v_2, v_3$ such that the $b$-path between $v_1$ and $v_2$, and the $b$-path from $v_3$ to $t$ does not touch any other node in $C_1$. By definition of $C_1$, every component before $C_1$ is touched at most once, thus we will find an $a$-path from $s$ to $C_1$ which is node-disjoint from the $b$-path (as the $b$-path does not use nodes in $A$ and touches every component before $C_1$ at most once, it follows from the definition of components).

Let now $s_{C_1}$ be the first node in $C_1$ which is seen by the $a$-path from $s$ to $x_1$. (Note: $s_{C_1}$ is unique.) We add a new node $s'$ connected to $s_{C_1}$ and $v_3$ and a new node $t'$ connected to $v_1$ and $v_2$. Clearly, $s'$ and $t'$ belong to the component, so there are two node-disjoint paths from $s'$ to $t'$ due to Menger’s theorem, see Theorem 8.2. Thus there are two node-disjoint paths, one from $s_{C_1}$ to $v_1$ or $v_2$ and one from $v_3$ to the other node ($v_2$ or $v_1$) in $G_a$.

As there are $b$-paths from $v_1$ to $v_2$ and from $v_3$ to $t$, and those are undirected, we can combine them with the node-disjoint $a$-paths and the $a$-path from $s$ to $s_{C_1}$ to obtain a solution.

We now turn to case 2, in which $C_1$ is touched exactly twice. Then there is another 2-connected component of $G_a$ from $s$ to $x_1$ that is touched at least twice. Let $C_2$, with $C_1 \neq C_2$, be the next 2-connected component of $G_a$ from $s$ to $x_1$ that is touched at least twice.

Since $C_1$ and $C_2$ are the 2-connected components closest to $s$ which are touched at least twice, we can find an $a$-path from $s$ to $C_1$ and one from $C_1$ to $C_2$. Especially, these $a$-paths can be chosen such that they are node-disjoint with the $b$-path because the $b$-path does not use nodes in $A$ and by definition of 2-connectedness. Let $v$ be the last touch point of the $b$-path in $C_1$ or $C_2$, closest to $t$. Since $v$ leads to $t$, we must use it last. If $v$ is in $C_2$, we can use the $b$-path in $C_1$ (it must exist, because $C_1$ is touched exactly twice), and then use an
a-path to \(v\) in \(C_2\). So we can assume without loss of generality that, \(v \in C_1\). Let \(u\) be the other touch point in \(C_1\). We name the first node of \(C_1\) in \(G_a\ \(s_1\) and the last node of \(C_1\) in \(G_a\ \(t_1\). (Note that \(s_1, t_1 \in A\) by definition of 2-connected component and 1-cut.) As we have a 2-connected component, we can conclude with Menger’s theorem, see Theorem 8.2, that there are two node-disjoint paths, one from \(s_1\) to \(t_1\) or \(u\), and one from \(v\) to \(u\) or \(t_1\). More precisely, we add a new node \(s'\) connected to \(s_1\) and \(v\) and a new node \(t'\) connected to \(u\) and \(t_1\). Then Menger’s theorem implies two node-disjoint paths from \(s'\) to \(t'\) in \(G_a\).

We explain how to construct a simple path matching \(a^*b^+a^+b^*\) from \(s_1\) to \(v\): If there exist node-disjoint paths from \(s_1\) to \(u\) and from \(t_1\) to \(v\), we can use the first one, then the \(b\)-path to the first node in \(C_2\), and from that node in \(C_2\) we can use a path to \(t_1\) and from there the node-disjoint one to \(v\). Otherwise, we have node-disjoint paths from \(s\) to \(t_1\) and from \(u\) to \(v\). We first use the path from \(s\) to \(t_1\) to go to the second component. As \(C_2\) is touched by the \(b\)-path, we go to one of these nodes and use the \(b\)-path from there to \(u\), then the \(a\)-path from \(u\) to \(v\).

We can then construct a solution by adding the \(a\)-path from \(s\) to \(s_1\) and the \(b\)-path from \(v\) to \(t\). The \(b\)-paths are node-disjoint since \(p\) was a simple \(b\)-path, and the \(a\)-paths are node-disjoint by definition of components. This concludes the proof.

By symmetry, the next result follows immediately for components of \(G_b\):

**Lemma 10.2.** Let \(C_1\) be 2-connected component of \(G_b\) from \(x_3\) to \(t\) and \(p\) be a simple \(a\)-path starting in \(s\) in \(G_a - B\) which

1. touches \(C_1\) in at least 3 nodes, or
2. touches \(C_1\) and another 2-connected component \(C_2\) of \(G_b\) from \(x_3\) to \(t\) both at least twice.

Then there is a simple path matching \(a^*b^+a^+b^*\) from \(s\) to \(t\).

We now prove that if Algorithm 2 returns “yes”, then we find paths \(p_{a1}, p_{a2}, p_{b1}, p_{b2}\) such that if \(p_{a1}\) and \(p_{b2}\) intersect, then \(p_{b2}\) touches each component of \(G_a\) in which they intersect at least twice. For a set of nodes \(S\), we call a path \(S\)-avoiding if none of the edges in the path uses a node in \(S\).

**Lemma 10.3.** If there are

- node-disjoint \(B\)-avoiding \(a\)-paths \(p_{a1}\) from \(s\) to \(x_1\) and \(p_{a2}\) from \(x_2\) to \(x_3\) and
- node-disjoint \(A\)-avoiding \(b\)-paths \(p_{b1}\) from \(x_1\) to \(x_2\) and \(p_{b2}\) from \(x_3\) to \(t\),

then there exist nodes \(y_1, y_2, y_3\) and

- node-disjoint \(B\)-avoiding \(a\)-paths \(p'_{a1}\) from \(s\) to \(y_1\) and \(p'_{a2}\) from \(y_2\) to \(y_3\) and
- node-disjoint \(A\)-avoiding \(b\)-paths \(p'_{b1}\) from \(y_1\) to \(y_2\) and \(p'_{b2}\) from \(y_3\) to \(t\),

such that, additionally, \(p'_{a1}\) is node-disjoint from \(p'_{b2}\) in every component in \(G_a\) from \(s\) to \(y_1\) that \(p'_{b2}\) touches at most once.

**Proof Sketch.** If \(p_{a1}\) and \(p_{b2}\) intersect in a 2-connected component of \(G_a\), then we can re-route \(p_{a1}\) because of the 2-connectedness. If \(p_{b2}\) touches this component only once, then the re-routed subpath of \(p_{a1}\) is node-disjoint from \(p_{b2}\), but the re-routed path might not be node-disjoint with \(p_{b1}\) or \(p_{b2}\). We explain how to reroute in these cases (possibly changing \(x_1, x_2, x_3\)). An example is shown in Figure 9. \(\square\)

**Proof.** Firstly, we can assume without loss of generality that all paths but \(p_{a1}\) and \(p_{b2}\) are pairwise node-disjoint (otherwise we shortcut and update \(x_1, x_2, x_3, A\), and \(B\) accordingly).
Let $C$ be the component of $G_a$ from $s$ to $x_1$ that is closest to $s$ such that $p_{a1}$ and $p_{b2}$ intersect and $p_{b2}$ touches $C$ at most once. We name the start and end of $C$ in $G_a$ $s_C$ and $t_C$, respectively. We name the intersection point of $p_{a1}$ and $p_{b2}$ in $C$ $x$.

Let $p'$ be a rerouting of $p_{a1}$ in $C$ which avoids $x$. Since $p_{b2}$ touches $C$ at most once, $p'$ is node-disjoint from $p_{b2}$. If it is node-disjoint from $p_{b1}$ and $p_{a2}$, we can continue with the next component contradicting the lemma. Otherwise, we perform a case distinction depending on whether $p_{b1}$ (case 1) or $p_{a2}$ (case 2) intersects with $p'$ first.

Case 1: Let $v$ denote the first intersection of $p'$ with $p_{b1}$. We can then shortcut to the simple path $p'[s, v] \cdot p_{b1}[v, x_2] \cdot p_{a2} \cdot p_{b2}$. As there is no 2-connected component after $C$ in $G_a$, we are done. The result follows with $y_1 = v, y_2 = x_2, y_3 = x, p'_{a1} = p'[s, v], p'_{b1} = p_{b1}[v, x_2], p'_{a2} = p_{a2}$ and $p'_{b2} = p_{b2}$.

Case 2: We show that we either find an alternative $a$-path which does not touch $p_{b2}$, or such that we can reroute as in Figure 9. For this rerouting to work we have to show that there is an $a$-path $p''$ from $s_C$ to the intersection with $p_{b2}$ which is node-disjoint from $p_{a1}[x, t_C]$:

So let us assume that $p'$ and $p[x, t_C]$ are not node-disjoint. We show how to find a better choice $p''$. Since $p'$ and $p[x, t_C]$ are not node-disjoint, there exists some node $u$ in $p'$ and $p[x, t_C]$, such that $p'[s_C, u]$ and $p[x, t_C]$ share only the node $u$. If $p'[s_C, u]$ is disjoint from $p_{b2}$, we are done with this specific component $C$, since we can use the path $p'' = p'[s_C, u] \cdot p_{a1}[u, t_C]$ which is disjoint from $p_{b2}$ to reroute in $C$, taking care of the shortcuts from the $p_{b1}$ path as in case 1 if necessary.

If $p'[s_C, u]$ is not disjoint from $p_{b2}$, then we can reroute as depicted in Figure 9. Let $y$ be the node where $p'[s_C, v]$ first touches $p_{a2}$. Note that $y \neq s_C$ and $y \neq u$ since $p_{a1}$ and $p_{b2}$ are disjoint by assumption. We can choose $y_1 = x_2, y_2 = x_1, y_3 = x$ and

$$p'_{a1} = p_{a1}[s, s_C] \cdot p'[s_C, y] \cdot (p_{a2}[x_2, y])^{rev},$$

$$p'_{b1} = (p_{b1})^{rev},$$

$$p'_{a2} = (p_{a1}[x, x_1])^{rev},$$

$$p'_{b2} = p_{b2}[x, t].$$

Here, $p^{rev}$ denotes the path $p$ read from right to left.

By construction, the paths $p'_{a1}, p'_{a2}$, and $p'_{b1}$ are node-disjoint. Furthermore, since $p_{a2}$ and $p_{b2}$ were node-disjoint, we do not have any more intersections of $p'_{a1}$ and $p'_{b2}$ after $C$. Since $C$ was the component closest to $s$ violating the lemma and our new paths starting from $C$, that is, $p'_{a1}[s_C, y_1]$ and $p'_{b2}$ are node-disjoint, the result follows.

Combining the lemmas so far, and rerouting again if necessary, we obtain:

**Lemma 10.4.** If there are

- node-disjoint $B$-avoiding $a$-paths $p_{a1}$ from $s$ to $x_1$ and $p_{a2}$ from $x_2$ to $x_3$ and
- node-disjoint $A$-avoiding $b$-paths $p_{b1}$ from $x_1$ to $x_2$ and $p_{b2}$ from $x_3$ to $t$,

then there exist nodes $y_1, y_2, y_3$ and

- node-disjoint $B$-avoiding $a$-paths $p'_{a1}$ from $s$ to $y_1$ and $p'_{a2}$ from $y_2$ to $y_3$ and
- node-disjoint $A$-avoiding $b$-paths $p'_{b1}$ from $y_1$ to $y_2$ and $p'_{b2}$ from $y_3$ to $t$,

such that additionally $p'_{a1}$ and $p'_{b2}$ share at most one node.

**Proof.** Using Lemma 10.3 we know that we can find paths $p_{a1}, p_{a2}, p_{b1}, p_{b2}$ such that $p_{a1}$ and $p_{b2}$ can only intersect in components which are touched more than once by $p_{b2}$. By
ALGORITHM 2: Deciding \textsc{USimPath}(a^*b^+a^+b^*)

\textbf{Input:} Undirected multigraph \(G = (V,E,u\text{edge})\), nodes \(s,t\)
\textbf{Output:} “Yes” if there is a simple path from \(s\) to \(t\) in \(G\) that matches \(a^*b^+a^+b^*\); “no” otherwise

\begin{verbatim}
for all nodes \(x_1,x_2,x_3\) do
  \(A \leftarrow \emptyset\) \quad \triangleright \quad \text{nodes exclusively for } a\text{-paths}
  \(B \leftarrow \emptyset\) \quad \triangleright \quad \text{nodes exclusively for } b\text{-paths}
  foreach 1-cut \(v\) between \(s\) and \(x_1\) in \(G_a\) do
    \[A \leftarrow A \cup \{v\}\]
  foreach 1-cut \(v\) between \(x_3\) and \(t\) in \(G_b\) do
    if \(v \in A\) then
      continue with next triple of \(x_1,x_2,x_3\)
    \(B \leftarrow B \cup \{v\}\)
  if there exist two node-disjoint \(a\text{-paths}\) between \((s,x_1)\) and \((x_2,x_3)\) that do not use nodes in \(B\),
  and there exist two node-disjoint \(b\text{-paths}\) between \((x_1,x_2)\) and \((x_3,t)\) that do not use nodes in \(A\) then
    return “yes”
return “no”
\end{verbatim}

Lemma 10.1 at most one such component can exist. (Otherwise there is a solution, and a solution implies node-disjoint simple paths \(p_{a1}\) and \(p_{b2}\).) So we only need to show that if \(p_{a1}\) and \(p_{b2}\) meet in an component \(C\) of \(G_a\) from \(s\) to \(x_1\) more than once, then we can reroute in \(C_1\). Again by Lemma 10.1, if \(p_{a1}\) and \(p_{b2}\) intersect (at least) three times in \(C\), we are done (as then \(p_{b2}\) touches \(C\) at least three times, so there is a solution and a solution implies node-disjoint simple paths \(p_{a1}\) and \(p_{b2}\)).

So let \(x\) and \(x'\) be the intersections of \(p_{a1}\) and \(p_{b2}\) in \(C\). For the proof, we will reroute the \(a\text{-path}\) in \(C_1\) in order to not use \(x'\). If the rerouted path \(p'_{a1}\) touches \(p_{b2}\) in any other node than \(x\), we are done by Lemma 10.1. If the rerouted path touches \(p_{b1}\) or \(p_{a2}\), we reroute as in Lemma 10.3 to ensure that \(p'_{a1}\) is node-disjoint from both \(p_{b1}\) and \(p_{a2}\). This completes the proof. \(\square\)

In Lemma 10.5 we can thus focus on paths where \(p_{a1}\) and \(p_{b2}\) share at most a single node.

**Lemma 10.5.** Algorithm 2 is correct.

**Proof.** If a solution exists, the algorithm will clearly return “yes”. So it remains to show: If the algorithm returns “yes”, then there exists a solution. By Lemma 10.4 we can then find such paths where \(p_{a1}\) and \(p_{b2}\) intersect in at most one node \(x\). Since \(p_{a1}\) does not use nodes in \(B\) and \(p_{b2}\) does not use nodes in \(A\), we have \(x \notin A \land B\). This implies that there is a 2-connected component \(C_1\) of \(G_a\) from \(s\) to \(x_1\) with \(x \in C_1\) and a 2-connected component \(C_2\) of \(G_b\) from \(x_3\) to \(t\) with \(x \in C_2\).

Let now \(p'_{a1}\) be a rerouting of \(p_{a1}\) in \(C_1\) and \(p'_{b2}\) be a rerouting of \(p_{b2}\) in \(C_2\). Let \(x'_a\) be the intersection of \(p'_{a1}\) with \(p_{b2}\) and \(x'_b\) be the intersection of \(p_{a1}\) with \(p'_{b2}\). (These nodes \(x'_a\) and \(x'_b\) must exist, since otherwise we would be done.) Note that \(x'_a \neq x'_b\) since otherwise \(p_{a1}\) and \(p_{b2}\) would both contain the node \(x'_a = x'_b\), which contradicts our choice of \(p_{a1}\) and \(p_{b2}\) (by Lemma 10.4 they share at most one node, which is \(x\)). Furthermore, since \(x_a \in C_1\) and \(x'_b \in C_2\), the intersections must be unique—otherwise we can use Lemma 10.1 or 10.2
to ensure that there is a solution. Using the same argument, it follows that \(x'_a \notin C_2\) and \(x'_b \notin C_1\). So the touch points must either be “before” or “after” \(C_1\) and \(C_2\).

Thus, those intersections then look (up to symmetry, that is, choice of \(s\) and \(t\)) like in Figure 10. We note that \(sC_1, tC_1, sC_2,\) and \(tC_2\) depend on \(s\) and \(t\) and have therefore been renamed \(uC_1, vC_1, uC_2,\) and \(vC_2\) where \(uC_1 = sC_1\) if \(s = s_1\), and \(uC_1 = tC_1\) otherwise. Analogously, \(uC_2 = sC_2\) if \(t = t_2\), and \(uC_2 = tC_2\) otherwise. We note that reroutings in \(C_1\) or \(C_2\) are allowed to use nodes in \(A\) and \(B\). Yet we can find for each choice of \(t \in \{t_1, t_2\}\) and \(s \in \{s_1, s_2\}\) a a simple path matching \(a^*b + a + b^*\) from \(s\) to \(t\) in Figure 10 as follows:

- If \(s = s_1, t = t_1\), then a possible solution in Figure 10 is a concatenation of an \(a\)-path from \(s_1\) to \(x\), a \(b\)-path from \(x\) via \(vC_2\) to \(x'_b\), an \(a\)-path from \(x'_b\) to \(x'_a\), and the \(b\)-path from \(x'_a\) to \(t_1\). More formally, we use

\[
p[s, x] \cdot (p[sC_2, x])^{rev} \cdot p_{b2}[sC_2, x'_b] \cdot (p[aTc_1, x'_b])^{rev} \cdot (p[a][x'_a, tC_1])^{rev} \cdot p_{a2}[x'_a, t].
\]

- If \(s = s_1, t = t_2\), then a possible solution in Figure 10 is a concatenation of an \(a\)-path from \(s_2\) to \(x'_b\), a \(b\)-path from \(x'_b\) via \(vC_2\) to \(x\), an \(a\)-path from \(x\) to \(x'_a\), and the \(b\)-path from \(x'_a\) to \(t_1\). More formally, we use

\[
p[s, x] \cdot (p[x'_a, x])^{rev} \cdot p_{a1}[x'_a, tC_1] \cdot p[a][tC_1, x'_b] \cdot p_{b2}[x'_b, tC_2] \cdot p_{b2}[tC_2, t].
\]

- If \(s = s_2, t = t_1\), then a possible solution in Figure 10 is a concatenation of an \(a\)-path from \(s_2\) to \(x'_b\), an \(b\)-path from \(x'_b\) via \(vC_2\) to \(x\), an \(a\)-path from \(x\) to \(x'_a\), and a \(b\)-path from \(x'_a\) to \(t_1\). More formally, we use

\[
p[a][s, x'_b] \cdot (p_{b2}[sC_2, x'_b])^{rev} \cdot p_{b2}[sC_2, x] \cdot p[a][tC_1, x'_b] \cdot (p[a][x'_a, tC_1])^{rev} \cdot p_{b2}[x'_a, t].
\]

- If \(s = s_2, t = t_2\), then a possible solution in Figure 10 is a concatenation of an \(a\)-path from \(s_2\) to \(x'_b\), a \(b\)-path from \(x'_b\) via \(vC_2\) to \(x'_a\), an \(a\)-path from \(x'_a\) to \(x\), and an \(b\)-path from \(x\) to \(t_2\). More formally, we use

\[
p[a][s, x'_b] \cdot (p_{b2}[sC_2, x'_b])^{rev} \cdot p_{b2}[sC_2, x] \cdot p[a][tC_1, x'_b] \cdot (p[a][x'_a, tC_1])^{rev} \cdot p_{b2}[x, t].
\]

We especially note that since the paths between \(s_1\) and \(uC_1, t_1\) and \(x'_a, x'_b\) and \(s_2,\) and \(vC_2\) and \(t_2\) are pairwise node-disjoint, the solution is independent of paths not depicted in Figure 10 (such as \(p_{b1}\) and \(p_{a2}\)).

\[\square\]

**Figure 9.** Possible rerouting in Lemma 10.3. The \(a\)-paths are blue and thick. We show that if the paths \(p_{a1}\) and \(p_{b2}\) touch each other at most once in a component, then we can reroute, even if this means rerouting \(p_{a2}\).

We note that we see no “easy” way to extend Theorem 9.7 to USimPath\( (a^*b^k a^*b^*), \) because the re-routing arguments of Lemmas 10.1 and 10.2 do not work for \(k \geq 2\).
Figure 10. The paths $p_a$ and $p_b$ with reroutings intersecting one another. Note that $s \in \{s_1, s_2\}$, $t \in \{t_1, t_2\}$, $\{u_{C_1}, v_{C_1}\} = \{s_{C_1}, t_{C_1}\}$, and $\{u_{C_2}, v_{C_2}\} = \{s_{C_2}, t_{C_2}\}$. The $a$-path is blue and thick, the $b$-path is black. The paths only intersect in the depicted nodes, the reroutings of $p_a$ and $p_b$ are dashed.

10.0.1. More than Two Alphabet Symbols. Theorem 7.3 generalizes in the following sense to larger alphabets, with only minor changes to the proof.

**Theorem 10.6.** $\text{USimPath}(L)$ is in P
(a) for every language definable by an SCRE($a, a^*, a^*, a^*$) and
(b) for every language definable by an SCRE($a, a^?, a^*, b, b^*, b^*, b^*$).

**Proof.** The proof is closely related to the proof of Theorem 7.3: The only changes are to use in (a) $r_i \in \text{SCRE}(a, a, a^*, a^*)$ instead of $r_i \in \text{SCRE}(a, a^?, a^*, a^*)$, and in (b) $r_i \in \text{SCRE}(a, a^?, b, b^*, b^*)$ instead of $r_i \in \text{SCRE}(a, a^?, b, b^*)$.

Whereas $\text{USimPath}(a^*ba^*)$ is tractable (Theorem 7.3), it follows from Theorem 5.2 that the following closely related language is intractable.

**Corollary 10.7.** $\text{USimPath}(a^*bc^*)$ is NP-complete.

This implies that SCRE($a^*, b, c^*$) can define languages for which $\text{USimPath}(L)$ is NP-hard and therefore shows that the SCREs used in Theorem 10.6 cannot be further extended without introducing languages for which $\text{USimPath}$ becomes intractable.

11. Parity Languages

We now discuss another interesting difference between directed and undirected multigraphs. Whereas $\text{Mod-2-path}$ is NP-complete for directed graphs [LP84], the problem is in P for undirected multigraphs [LP84]. We generalize this tractability result to a wide class of languages involving parity tests.

Assume that $\Sigma = \{a_1, \ldots, a_\ell\}$. The Parikh vector of a word $w$ is defined as $p(w) = (|w|_{a_1}, \ldots, |w|_{a_\ell})$, where $|w|_{a_i}$ is the number of occurrences of the label $a_i$ in $w$. The Parikh image of a language $L$ is the set $\{p(w) \mid w \in L\}$. A parity set is a semi-linear set of the form $\{v_1 + v_2n \mid n \in \mathbb{N}, v_1 \in V_1\}$, where $V_1 \subseteq \{0, 1\}^\ell$ and $v_2 = (2, \ldots, 2) \in \{2\}^\ell$. A parity language is a language for which its Parikh image is a parity set. Every such language is regular.

**Theorem 11.1.** $\text{UTrail}(L)$ and $\text{USimPath}(L)$ are in P for every parity language $L$. 

---

**Figure 10.** The paths $p_a$ and $p_b$ with reroutings intersecting one another. Note that $s \in \{s_1, s_2\}$, $t \in \{t_1, t_2\}$, $\{u_{C_1}, v_{C_1}\} = \{s_{C_1}, t_{C_1}\}$, and $\{u_{C_2}, v_{C_2}\} = \{s_{C_2}, t_{C_2}\}$. The $a$-path is blue and thick, the $b$-path is black. The paths only intersect in the depicted nodes, the reroutings of $p_a$ and $p_b$ are dashed.
As the proof of this theorem relies on the minor theorem on group-labeled graphs [Huy09], we start with some background: Let $\Gamma$ be a finite abelian group. A $\Gamma$-labeled graph is a directed graph whose edges are labeled with a group-label in $\Gamma$. The group-label of an edge $e$ is denoted by $\gamma_G(e)$. Following an edge in its direction adds the value $\gamma_G(e)$, while following it in the reverse direction adds the value $-\gamma_G(e)$. The group-value of a path is the sum of the values of its edges. A $\Gamma$-labeled graph $H = (V_H, E_H, \mathcal{E}_H)$ is a minor of a $\Gamma$-labeled graph $G = (V_G, E_G, \mathcal{E}_G)$ if and only if $V_H \subseteq V_G$ and for each edge $e$ in $H$, there exists a simple path $p_e$ with value $\gamma_H(e)$ from origin$(e)$ to destination$(e)$ in $G$. Furthermore, except for their first and last node, all $p_e$ are pairwise node-disjoint.

Huynh [Huy09] proves that for any fixed $\Gamma$-labeled graph $H$, there is a polynomial time algorithm which determines if an input $\Gamma$-labeled graph $G$ contains a minor isomorphic to $H$. We want to use this to solve some group-labeled variant of the kDisjointPaths problem, that is, given a group-labeled graph $G$, pairs of nodes $(s_i, t_i)_{i \in [k]}$ and values $(\gamma_i)_{i \in [k]}$, we want to know if there are simple paths $p_i$ from $s_i$ to $t_i$ with group value $\gamma_i$ in $G$ such that the $p_i$ are pairwise node-disjoint. However, Kawase et al. [KKY20, Footnote 3] observed that the reduction in [Huy09] cannot distinguish between two paths, one from $s_j$ to $t_j$ and one from $s_i$ to $t_i$, and two paths, one from $s_i$ to $s_j$ and one from $t_i$ to $t_j$ for any distinct $i$ and $j$. Angela Bonifati and Guillaume Bagan [BB19] pointed out that this can be fixed by considering $\Gamma \times \mathbb{Z}_3 \times \cdots \times \mathbb{Z}_3$-labeled graphs instead. In these graphs, every edge label can be interpreted as a vector of length $k + 1$. Let $G'$ be the graph obtained from $G$ by relabeling all edges $e$ with $(\gamma_G(e), 0, \ldots, 0)$ and adding $2k$ new nodes $(s'_j, t'_j)_{j \in [k]}$ and edges from $s'_i$ to $s_i$ and from $t_i$ to $t'_i$ which are labeled with the $\epsilon_{i+1}^{th}$ unit vector over $\mathbb{Z}_{k+1}$, that is, $(0, \ldots, 0, 1, 0, \ldots, 0)$ where the $i+1$th entry is $1$ and all others are $0$. Since the only simple path with group-label $(\gamma_i, 0, \ldots, 0, 2, 0, \ldots, 0)$, where the $i+1$th entry is $2$, is from $s'_i$ to $t'_i$, this implies: There are simple paths $p'_i$ with group value $(\gamma_i, 0, \ldots, 0, 2, 0, \ldots, 0)$ which are pairwise node-disjoint in $G'$ if and only if there are simple paths $p_i$ from $s_i$ to $t_i$ with group value $\gamma_i$ in $G$ such that the $p_i$ are pairwise node-disjoint.

In fact, the fix of Bonifati and Bagan inspired us to the following proof.

Proof of Theorem 11.1. We start with the proof for simple paths. The proof idea is to interpret $L$ as a language over a finite abelian group for which group-labeled and undirected graphs are closely related. We then relate the problem of finding a simple path in an undirected multigraph to the problem of finding a minor in the group-labeled graph, which can be decided in polynomial time, see Huynh [Huy09].

In order to make group-labeled graphs coincide with undirected graphs, we need that $\gamma_G(e) = -\gamma_G(e)$ for every edge $e$. This is the case if every element of the group is its own inverse, that is, when there is only a single element or the group is $\mathbb{Z}_2 = \{0, 1\}$ or a direct product thereof, that is, of the form $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \cdots \times \mathbb{Z}_2$. Every parity language can be interpreted as a language over a finite abelian group of the form $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \cdots \times \mathbb{Z}_2$. For instance, if $\Sigma = \{a, b\}$, then an $a$-edge can be encoded as $(1, 0)$, a $b$-edge as $(0, 1)$. Conversely, if the value of a path in a so-constructed, group-labeled graph is $(0, 1)$, we can interpret it as a path with an even number of $a$’s and odd number of $b$’s. In order to find a simple path from $s$ to $t$ that matches $L$ in an undirected multigraph $G$ (if it exists) we will:

1. We construct the underlying undirected graph $G_{\text{graph}}$ of $G$, that is, if there are edges $e_1, e_2$ in $G$ with $\mathcal{E}(e_1) = \mathcal{E}(e_2)$, then we remove one of them and repeat until $\mathcal{E}$ is an injective function.
We construct a group-labeled graph $G'$ that is obtained from $G_{\text{graph}}$ by relabeling the edges: We replace $a_i$ with the $i^{th}$ unit vector over $\mathbb{Z}_2^k$, which is $(0,\ldots,0,1,0,\ldots,0)$.

Furthermore, we direct each edge in an arbitrary direction.

We define a set of group-labeled graphs $\mathcal{H}$, which are edges from $s$ to $t$ that are labeled with a vector in $V_1$. (Where $V_1$ comes from the Parikh image of $L$.)

We then test if there is a graph $H \in \mathcal{H}$ that is a minor of $G'$. If such a minor exists, then there is a simple path from $s$ to $t$ in $G$ that matches $L$, otherwise there is none.

This procedure can be conducted in polynomial time since all changes to $G$ are linear in $|G| + k$, the graphs in $\mathcal{H}$ depend only on $L$ and are therefore of fixed size, and we can test for $|V_1|$ minors $G'$ using the polynomial time algorithm proposed by Huynh [Huy09].

Towards correctness, we first observe that a simple path in an undirected multigraph $G$ exists if and only if that simple path exists in its underlying undirected graph $G_{\text{graph}}$, since a simple path can use at most one edge between each pair of nodes. By construction of $G'$ and since $L$ is a parity language, a simple path from $s$ to $t$ in $G_{\text{graph}}$ that matches $L$ is a simple path from $s$ to $t$ in $G'$ whose group-value is in $V_1$. We can test for the latter using $|V_1|$ minor tests, which completes the correctness proof.

For trails, we proceed similar to the simple path case. That is, we again interpret $L$ as a language over a finite abelian group of the form $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \cdots \times \mathbb{Z}_2$ and relabel the edges of the multigraph $G$ accordingly, that is, we replace $a_i$ with the $i^{th}$ unit vector over $\mathbb{Z}_2^k$, which is $(0,\ldots,0,1,0,\ldots,0)$, and direct the edges arbitrarily.

We then relate the problem to the case for simple paths by using some variant of the extended line graph, introduced in Section 2.6. More precisely, we replace every node $v$ by a clique in which every edge of the clique is labeled $(0,\ldots,0)$. The size of the clique replacing $v$ is the number of edges adjacent to $v$. Similar to Lemma 2.9, if $v$ is the start or end-node of the trail, we add extra nodes $s^*$ or $t^*$ to the newly created clique.

More formally, let $G' = (V', E', E')$ be the group-labeled multigraph obtained from $G$ by relabeling the edges with unit vectors, and let nodes $s$ and $t$ be given. We define the group-labeled graph $G^* = (V^*, E^*, E^*)$ and $s^*, t^*$ as follows:

\[
V^* = \{(v,e) \mid v \in V', e \in E', e \text{ is incident to } v\} \cup \{s^*, t^*\}
\]

\[
E^* = \{((v,e_1),(0,\ldots,0),(v,e_2)) \mid (v,e_1),(v,e_2) \in V^*\}
\]

\[
\cup\{(v_1,e),\text{lab}(e),(v_2,e)) \mid (v_1,e),(v_2,e) \in V^*\}
\]

\[
\cup\{(s^*,(0,\ldots,0),(v,e)) \mid s = v,(v,e) \in V^*\}
\]

\[
\cup\{(t^*,(0,\ldots,0),(v,e)) \mid s = v,(v,e) \in V^*\},
\]

and $E^*((x,\sigma,y)) = \{(x,\sigma,y) \mid (x,\sigma,y) \in E^*\}$.

By construction, there is a trail from $s$ to $t$ in $G$ matching $L$ if and only if there is a simple path from $s^*$ to $t^*$ in $G^*$ whose group-value is in $V_1$. The latter can again be tested by constructing a set of group-labeled graphs from $V_1$ and testing if $G^*$ has one of them as a minor.

Since Huynh [Huy09] can also deal with more complex minors, (including minors which are $k$ disjoint edges,) it follows that:

**Lemma 11.2.** Let $L_1, \ldots, L_k$ be parity languages over $\Sigma$. Then we can find $k$ node-disjoint simple paths (or $k$ edge-disjoint trails) from $s_i$ to $t_i$ matching $L_i$ in polynomial time.

**Proof.** Since $L_1, \ldots, L_k$ are parity languages, there exist sets $V_{1,1}, \ldots, V_{1,k} \subseteq \{0,1\}^\ell$ such that each $L_i$ can be written in the form $\{v_1 + v_2n \mid n \in \mathbb{N}, v_1 \in V_{1,i}\}$ and $v_2 = (2,\ldots,2)$.
with \( i \in |k| \). The only change to the proof of Theorem 11.1 is that the group-labeled graphs in \( \mathcal{H} \) are not single edges, but \( k \) node-disjoint edges. More precisely, a group-labeled graph in \( \mathcal{H} \) has for each \( i \in |k| \) an edge from \( s_i \) to \( t_i \) that is labeled with a word in \( V_{1,i} \). Since each possible combination is in \( \mathcal{H} \), \( |\mathcal{H}| = |V_{1,1}| \cdots |V_{1,k}| \).

Remark 11.3. For simple paths it is important that \( L_1, \ldots, L_k \) use the same alphabet, as the problem of finding two node-disjoint paths, one labeled \( a^* \), the other \( b^* \) is NP-complete, see Gourves et al. [GdLMM12, Theorem 16].

Corollary 11.4. Let \( L_1, \ldots, L_k \) be parity languages over \( \Sigma \) and let \( F_1, \ldots, F_{k+1} \) be finite languages. Then \( \text{USimPath}(F_1 L_1 F_2 \cdots L_k F_{k+1}) \) and \( \text{UTrail}(F_1 L_1 F_2 \cdots L_k F_{k+1}) \) are in \( P \).

Proof. We enumerate the node-disjoint (or edge-disjoint) finite simple paths (or trails) \( (p_1, \ldots, p_{k+1}) \) matching \( F_1 \), \( F_{k+1} \). Let \( p_i \) be a path from \( t_{i-1} \) to \( s_i \) for each \( i \in |k+1| \). Then we can use Lemma 11.2 to test in polynomial time if there exist \( k \) node-disjoint simple paths (or edge-disjoint trails) from \( s_i \) to \( t_i \) matching \( L_i \) in the multigraph without the inner nodes (or edges) of the paths \( (p_1, \ldots, p_{k+1}) \).

Using the observation that \( G_a \) and \( G_b \) are edge-disjoint for all pair of symbols \( a \neq b \) we obtain:

Corollary 11.5. Let \( L_1, \ldots, L_k \) be parity languages over alphabets \( \Sigma_1, \ldots, \Sigma_n \), such that \( \Sigma_i = \Sigma_j \) or \( \Sigma_i \cap \Sigma_j = \emptyset \) for each pair \( i, j \). Let \( F_1, \ldots, F_{k+1} \) be finite languages. Then \( \text{UTrail}(F_1 L_1 F_2 \cdots L_k F_{k+1}) \) is in \( P \).

Proof. We explain how to decide \( \text{UTrail}(F_1 L_1 F_2 \cdots L_k F_{k+1}) \) in polynomial time. Let \( G \) be an undirected multigraph. We enumerate the edge-disjoint trails \( (p_1, \ldots, p_{k+1}) \) matching \( F_1, \ldots, F_{k+1} \) in \( G \). Let \( p_i \) be a path from \( t_{i-1} \) to \( s_i \) for each \( i \in |k+1| \). Let \( I_1, \ldots, I_m \) be sets of indices such that \( \Sigma_i = \Sigma_j \) for all \( i, j \in I_x \) and \( \Sigma_i \cap \Sigma_j = \emptyset \) for all \( i \in I_x, j \in I_y \) and \( x \neq y \). Since for each \( x \in |m| \) the subgraphs of \( G \) restricted to edges with labels in \( \Sigma_i \) with \( i \in I_x \) are edge-disjoint from each subgraph of \( G \) restricted to edges with labels in \( \Sigma_j \) with \( j \notin I_x \) per construction of \( I_1, \ldots, I_m \), we can perform tests on each subgraph separately. More precisely, for each \( x \in |m| \) we use Lemma 11.2 to test in polynomial time if there exist \( |I_j| \) edge-disjoint trails from \( s_i \) to \( t_i \) matching \( L_i \) with \( i \in I_x \) in the multigraph restricted to edges with labels in \( \Sigma_i \) with \( i \in I_x \), and without edges of the paths \( (p_1, \ldots, p_{k+1}) \).

12. Enumeration

Given a path \( p = e_1 \cdots e_n \) and \( 1 \leq i \leq j \leq n \), we denote by \( p[i, j] \) the subpath \( e_i \cdots e_j \).

We denote the set of edges of path \( p \) with \( E(p) = \{ e_1, \ldots, e_n \} \). For convenience, we define \( p[1, 0] = \varepsilon \) and therefore \( V(p[1, 0]) = E(p[1, 0]) = \emptyset \). Furthermore, let \( p \) be a path from \( s \) to \( t \).

We denote by \( \text{destination}(p[i, j]) \) end of the subpath \( p[i, j] \) and define \( \text{destination}(p[1, 0]) = s \), that is, the start of \( p \).

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 \( P(i) \). 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 \( P(i) \) 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.

For several enumeration problems, we will consider the *radix order* on paths. To this end, we assume that there exists an order $<$ on $\Sigma$. 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 $\text{lab}(p_1) < \text{lab}(p_2)$.

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 $f$.

**Enumeration Problems.** We now consider the following problems:

| EnumUSimPaths($L$) | Given: An undirected multigraph $G$, nodes $s$, $t$. | Question: Enumerate the simple paths from $s$ to $t$ in $G$ that match $L$. |
| EnumUTrails($L$) | Given: An undirected multigraph $G$, nodes $s$, $t$. | Question: Enumerate the trails from $s$ to $t$ in $G$ that match $L$. |

Martens et al. [MT19, MNT20] have shown that (variants of) Yen’s algorithm [Yen72] can be used to enumerate all simple paths or trails matching a regular expression in directed multigraphs. Since Yen’s original algorithm also works on undirected graphs, their results immediately carry over to undirected multigraphs. Yet Yen’s algorithm requires a shortest paths subroutine. But as the next proposition shows, there are languages in $\text{USP}^{\text{tract}}$ and $\text{UT}^{\text{tract}}$ for which no polynomial time algorithm for finding a shortest path is known.

**Proposition 12.1.** There is no known polynomial time algorithm that returns a shortest simple path (or trail) from $s$ to $t$ that matches $a^*bca^*$.

**Proof.** We will reduce the problem to the min-sum $k$ disjoint paths problem. This problem asks, given an undirected graph, nodes $s_1, t_1, \ldots, s_k, t_k$ for $k$ disjoint paths, one from $s_i$ to $t_i$ for each $i \in [k]$, such that the sum of their lengths is minimal. For each integer $k$, this problem comes in a node-disjoint and an edge-disjoint variant.

The complexity of min-sum $k$ disjoint paths problem for $k \geq 2$ has been open for several decades, although it has been extensively studied, see Fenner et al. [FLP16] for an overview of the edge-disjoint variant and Kobayashi and Sommer [KS10] for an overview of the node-disjoint variant.\footnote{We note that there exists a polynomial time Monte Carlo algorithm for the case $k = 2$, see Björklund and Husfeld [BH19], but no deterministic polynomial time algorithm is known. If there is only one start and one endnode, a polynomial time algorithm for min-sum $k$ node-/edge-disjoint paths with $k = 2$ was introduced by [Suu74, ST84] and later extended to arbitrary $k \geq 2$ by Bhandari [Bha97]. Yang and Zheng [YZ06] gave a polynomial time algorithm for the case $k = 2$ with $s_1 = s_2$ and $t_1 \neq t_2$.}
We now show that finding a shortest simple path from \( s \) to \( t \) that matches \( a^*bca^* \) is possible in polynomial time if and only if the \textit{min-sum two node-disjoint paths problem} can be solved in polynomial time.

- To return a shortest simple path from \( s \) to \( t \) that matches \( a^*bca^* \) in a undirected multigraph \( G \), we first iterate over all possible triples of pairwise different nodes \( x, y, z \). For each such triple, we test for a \( b \)-edge from \( x \) to \( y \) and a \( c \)-edge from \( y \) to \( z \). If this test succeeds, we only need to find two node-disjoint path in \( G_a \) without \( y \), one from \( s \) to \( x \) and one from \( z \) to \( t \), such that the sum of their lengths is minimal. Although \( G_a \) is a undirected multigraph, it suffices to consider its underlying undirected graph, since a shortest path will not use multiple edges between nodes and the node-disjointness ensures that the two different paths will not use an edge between the same pair of nodes. Thus this problem is exactly the min-sum two node-disjoint paths problem. After having iterated over all triples, we return the overall shortest path.

- On the other hand, we can find two node-disjoint paths from \( s_1 \) to \( t_1 \) and from \( s_2 \) to \( t_2 \) such that the sum of their length is minimal as follows: We relabel every edge with \( a \) and add a path labeled \( bc \) from \( t_1 \) to \( s_2 \). Then the problem reduces to finding a shortest simple path matching \( a^*bca^* \) from \( s_1 \) to \( t_2 \) and returning the respective subpaths.

We now turn to trail semantics. That is, we now show that finding a shortest trail from \( s \) to \( t \) that matches \( a^*bca^* \) is possible in polynomial time if and only if the \textit{min-sum two edge-disjoint paths problem} can be solved in polynomial time.

- To return a shortest trail from \( s \) to \( t \) that matches \( a^*bca^* \) in a directed multigraph \( G \), we first iterate over all possible pairs of nodes \( u, v \) and edges \( e_1, e_2 \) with \( \text{lab}(e_1) = b, \text{lab}(e_2) = c \) such that \( e_1 e_2 \) is a path from \( u \) to \( v \). Since \( G \) and therefore \( G_a \) is a multigraph, while the the min-sum two edge-disjoint paths problem is defined over undirected (simple) graphs, we cannot use \( G_a \) directly. To this end, we construct a simple graph \( G' \) by (1) removing all edges of the form \((z, a, z)\) from \( G_a \) and (2) replacing every edge with two new ones and an extra node. More formally, let \( G_a = (V, E_a, \mathcal{E}_a) \) be the subgraph of \( G \) restricted to \( a \)-edges. We define \( G' = (V', E', \mathcal{E}') \) with \( V' = V \cup \{z_e \mid e \in E_a\}, E' = \{e_1, e_2 \mid e \in E_a \text{ and } |\text{Node}(e)| > 1\}, \) and if \( \mathcal{E}_a(e) = (x, a, y) \), then \( \mathcal{E}'(e_1) = (x, a, z_e) \) and \( \mathcal{E}'(e_2) = (z_e, a, y) \). Now we only need to find two edge-disjoint paths in \( G' \), one from \( s \) to \( u \) and one from \( v \) to \( t \), such that the sum of their lengths is minimal, which is exactly the min-sum two edge-disjoint paths problem.

After having iterated over all pairs of nodes and edges, we return the overall shortest trail.

- On the other hand, we can find two edge-disjoint paths from \( s_1 \) to \( t_1 \) and from \( s_2 \) to \( t_2 \) such that the sum of their length is minimal as follows: We relabel every edge with \( a \) and add a path labeled \( bc \) from \( t_1 \) to \( s_2 \). Then the problem reduces to finding a shortest trail matching \( a^*bca^* \) from \( s_1 \) to \( t_2 \) and returning the respective subpaths.

This completes the proof. \( \square \)

We will therefore show that Yen’s algorithm can be adapted to work with subroutines that return a (not necessarily shortest) trail. Note that Martens and Trautner [MT19] showed the equivalent Theorem for simple paths.

While under simple path semantics one could simply remove the word \textit{shortest}, we have to be more careful under trail semantics. More precisely, we additionally need to test before returning a path \( p \) whether it has a prefix which was not yet written to the output. The
ALGORITHM 3: Yen’s algorithm changed to work with trails on multigraphs, see Martens et al. [MNT20, MNP21]

| Input: | Directed or undirected multigraph $G = (V, E, \mathcal{E})$, nodes $s, t \in V$, a regular language $L$ |
| Output: | All trails from $s$ to $t$ in $G$ that match $L$ under bag semantics |
| 1 | $A \leftarrow \emptyset$ \> $A$ is the set of trails already written to output |
| 2 | $B \leftarrow \emptyset$ \> $B$ is a set of trails from $s$ to $t$ matching $L$ |
| 3 | $p \leftarrow$ a shortest trail from $s$ to $t$ matching $L$ \> $p \leftarrow \bot$ if no such trail exists |
| 4 | while $p \neq \bot$ do |
| 5 | output $p$ |
| 6 | Add $p$ to $A$ |
| 7 | for $i = 0$ to $|p|$ do |
| 8 | $G' \leftarrow (V, E - E(p[1,i]), \mathcal{E}|_{E - \mathcal{E}(p[1,i])})$ \> Delete the edges of $p[1,i]$ |
| 9 | $S = \{e \in E \mid p[1,i] \cdot e$ is a prefix of a trail in $A\}$ |
| 10 | $p_1 \leftarrow$ a shortest trail from destination($p[1,i]$) to $t$ in $G'$ that matches $(\text{lab}(p[1,i]))^{-1}L \setminus \{\varepsilon\}$ |
| 11 | and does not start with an edge from $S$ |
| 12 | Add $p[1,i] \cdot p_1$ to $B$ \> $p \leftarrow \bot$ if $B = \emptyset$ |
| 13 | $p \leftarrow$ a shortest trail in $B$ |

reason herefore is that, if there are trails $p_1, p_2$ from $s$ to $t$ such that $p_1$ is a prefix of $p_2$ and if $p_2$ is first written to the output, then the “original” algorithm might not find $p_1$.

**Lemma 12.2.** Algorithm 3 is also correct if we make the following changes: (1) the assignments on lines 3 and 10 assign (possibly non-shortest) trails to $p$ and $p_2$, respectively, and (2) in line 12 it is additionally tested if $p$ contains a prefix $p'$ which is a trail from $s$ to $t$ that matches $L$ and is not yet in $A$. If this test succeeds, we take a shortest such prefix instead of $p$.

**Proof.** On line 8, Algorithm 3 considers the last path $p$ that was added to $A$ and compares it to every path $p_1$ in $A$ that shares the first $i$ edges with $p$. Since the algorithm forbids, for every such path $p_1$, the next edge to be $p_1[i + 1, i + 1]$, we cannot find a trail more than once. So we just have to show that every trail is eventually found.

Clearly, the algorithm is correct if there exists at most one trail from $s$ to $t$ that matches $L$. Assume, towards a contradiction, that there is more than one such trail and the algorithm terminated, that is, $B$ is empty, but there exists a trail $p'$ from $s$ to $t$ that matches $L$ and is not in $A$. Due to (2), $p'$ cannot be a prefix of a path already in $A$. Let $C$ be the set of paths in $A$ that share the longest prefix with $p'$. Since all paths in $A$ start in $s$ and $A \neq \emptyset$ due to line 3, we have that $C \neq \emptyset$. Let $i$ be the maximal integer with $p[1,i] = p'[1,i]$ for all $p \in C$. We note that $i < |p'|$ due to (2).

Let $p \in C$ be the last path that was added to $A$. After adding $p$ to $A$, we must have executed the for loops again. In line 8, we then cannot have deleted the edge $p'[i + 1, i + 1]$, otherwise this would contradict the definition of $C$. Since $p$ was the last path from $C$ that we added to $A$ and $B$ is empty by assumption, $p'$ must have been found during this execution of the while-loop. Thus we must have $p' \in B$ or $p' \in A$. Contradiction.

We now study the time guarantee of Algorithm 3. The time needed depends on the subroutines in lines 3 and 10. The next lemma implies that if there are polynomial time
algorithms (FPT algorithms, respectively) for the subroutines, then the enumeration is possible in polynomial delay (FPT delay, respectively).

Lemma 12.3. 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 an undirected multigraph \( G \), nodes \( s \) and \( t \), a word \( w \) with \( |w| \leq |G| \), and \( r \in \mathcal{R} \), return in time \( f(|G|,|r|) \) (with \( f(|G|,|r|) \geq |G| \)),

(1) a trail from \( s \) to \( t \) in \( G \) that matches \( L(r) \) if it exists and “no” otherwise and
(2) a path from \( s \) to \( t \) in \( G \) that matches \( w^{-1}L(r) \) if it exists and “no” otherwise respectively, then \( \text{EnumUTrails}(\mathcal{R}) \) is in delay \( O(|E|^2 \cdot f(|G|,|r|)) \) with preprocessing time \( O(f(|G|,|r|)) \).

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

Proof. We use the variant of Algorithm 3 described in the statement of Lemma 12.2, with calls to algorithm \( \mathcal{A}_1 \) on line 3, and up to \( |E| \) calls of algorithm \( \mathcal{A}_2 \) on line 10. Furthermore, we choose an arbitrary path, shortest path, or smallest path in radix order in \( B \) on line 12, depending on whether we want to enumerate in arbitrary order, order of increasing length, or radix order, respectively. The correctness for \( \text{EnumTrails}(\mathcal{R}) \) and \( \text{EnumUTrails}(\mathcal{R}) \) follows from Lemma 12.2.

Clearly, we need time \( O(f(|G|,|r|)) \) to output the first path (if it exists). Then, Algorithm 3 does up to \( |E| \) iterations in line 7. We note that all paths are trails and therefore have length at most \( |E| \). If we use a prefix tree as a data structure for \( A \), we can insert a path in time \( O(|E|^2) \) or find a path \( p \) in \( A \) in \( O(|E|) \) time. Thus we can also find the right node in the prefix tree and then compute \( S \) in line 9 in \( O(|E|) \) time. In line 10, we need multiple calls to \( \mathcal{A}_2 \) to ensure that the path starts with an edge not in \( S \). More precisely, let \( G' = (V', E', \mathcal{E}') \) for each edge \( e \in E' - S \), we compute a new multigraph \( G'_e = (V'_e, E'_e, \mathcal{E}'_e) \) by adding a new node \( s' \) to \( G' \), and change \( \mathcal{E}' \) such that \( \mathcal{E}'(e) = (s', \text{lab}(e), v) \) instead of \( \mathcal{E}'(e) = (\text{destination}(p[1,i]), \text{lab}(e), v) \). Then there is a trail from \( s' \) to \( t \) in \( G'_e \) that matches \( (\text{lab}(p[1,i]))^{-1}L \setminus \{e\} \) if and only if there is such a trail from \( \text{destination}(p[1,i]) \) to \( t \) in \( G' \) that starts with \( e \). Furthermore, these trails use the same edges. Thus we can call \( \mathcal{A}_2 \) on each \( G'_e \) with \( e \in E' - S \) and take an arbitrary, an overall shortest, or an overall smallest trail in radix order as \( p_1 \).

In line 12 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 (without the \( |p| \)), which is a minimal simple path. Finding and deleting are in time \( O(|E|) \). Thus, we have a delay of \( O(f(|G|,|r|)) \) until the first output, and afterwards delay \( O(|E||E| + |E| \cdot f(|G|,|r|)) \).

Since Yen’s algorithm also works on undirected graphs, the undirected version of Martens and Trautner [MT19, Lemma 8.4] follows:

Lemma 12.4. 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 an undirected multigraph \( G \), nodes \( s \) and \( t \), a word \( w \) with \( |w| \leq |G| \), and \( r \in \mathcal{R} \), return in time \( f(|G|,|r|) \) (with \( f(|G|,|r|) \geq |G| \)),

(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 \( \text{EnumUSimPaths}(\mathcal{R}) \) is in delay \( O(|V| \cdot f(|G|,|r|)) \) with preprocessing time \( O(f(|G|,|r|)) \).
Furthermore, if $A_1$ and $A_2$ always return a shortest simple path (respectively, a smallest simple path in radix order), then the enumeration can be done in order of increasing length (respectively, in radix order), with the same time guarantees.

We now show how to obtain algorithms $A_1$ and $A_2$ for Lemmas 12.3 and 12.4. More precisely, we use the standard method of self-reducibility to query a decision algorithm multiple times in order to reconstruct the solution. That is, each polynomial time algorithms that decide whether a simple path or trail matching $L$ exists in $G$ can be used to obtain a polynomial time algorithm that can return such a path (if it exists).

**Lemma 12.5.** Let $G = (V,E,\mathcal{L})$ be a directed or undirected multigraph, $s$ and $t$ two nodes, and $B$ be an algorithm that decides in time $x$ whether there is a simple path (respectively trail) from $s$ to $t$ in $G$ that matches $L$. Then there exists an algorithm that returns in time $O(|E| \cdot x)$ (respectively $O(|E|^2 \cdot x)$) a simple path (respectively trail) from $s$ to $t$ in $G$ that matches $L$ if one exists, and “no” otherwise.

**Proof.** For simple paths, we construct an algorithm $A$ as follows: If $B$ returns “no”, $A$ will also return “no”. Otherwise, there exists a simple path from $s$ to $t$ in $G$ which matches $L$. Let $e_1$ be an edge adjacent to $s$. Let $G'$ be the graph obtained from $G$ by removing all edges adjacent to $s$ except $e_1$. If $B$ returns “no”, we remove $e_1$ from $G$ and repeat the procedure with the next edge adjacent to $s$, otherwise, if $B$ returns “yes” on graph $G'$, we can choose $e_1$ as first edge and remove the other edges adjacent to $s$ (that is, except $e_1$) permanently from $G$. After we have found the $i$th edge, ending in a node $u$ with $u \neq t$, we can find the $(i+1)$th edge with a similar method: Let $e_{i+1}$ be an edge adjacent to $u$. Let $G'$ be the graph obtained from $G$ by removing all edges adjacent to $u$ except $e_i$ and $e_{i+1}$. If $B$ returns “no”, we remove $e_{i+1}$ from $G$ and repeat the procedure with the next edge adjacent to $u$, otherwise, if $B$ returns “yes” on graph $G'$, we can choose $e_{i+1}$ as $(i+1)$th edge and remove the other edges adjacent to $u$ (that is, except $e_i$ and $e_{i+1}$) permanently from $G$. Once we found an edge $e_t$ ending in $t$, we return $e_1 \cdots e_t$.

The so-constructed path $e_1 \cdots e_t$ is a solution by construction. Furthermore, since every edge is considered at most once, the running time is bounded by $O(|E| \cdot x)$.

We now turn to trail semantics. We construct algorithm $A$ as follows: If $B$ returns “no”, $A$ will also return “no”. Otherwise, there exists a trail from $s$ to $t$ which matches $L$.

We enumerate over all edges $e_1$ adjacent to $s$. Let $E(e_1) = (s,a,u)$. We define $G' = (V',E',\mathcal{L}')$ to be the undirected multigraph obtained from $G = (V,E,\mathcal{L})$ by adding a new node $s'$ and defining $E'(e_1) = (s',a,u)$. We now use $B$ to decide the existence of a trail from $s'$ to $t$ in $G'$ that matches $L$. If $B$ returns “no”, there is no solution which uses $e_1$ as first edge, so we continue with the next edge adjacent to $s$. If $B$ returns “yes”, then there is a solution which uses $e_1$ as first edge, and we permanently delete $e_i$ from $G$.

After we have found the $i$th edge, ending in a node $u$, we can find the $(i+1)$th edge with a similar method: Let $e_{i+1}$ be an edge adjacent to $u$ and $e_{i} \neq e_{i+1}$. Let $G'$ be the graph obtained from $G$ by removing $e_{i+1}$, and adding a new node $s'$ and a path labeled $\text{lab}(e_1 \cdots e_{i+1})$ from $s'$ to $u$. We use $B$ to decide if there is a trail from $s'$ to $t$ matching $L$ in $G'$. If it returns “no”, we repeat the procedure with the next edge adjacent to $u$, otherwise, if $B$ returns “yes” on graph $G'$, we can choose $e_{i+1}$ as $(i+1)$th edge and permanently remove $e_{i+1}$ from $G$. Once we found an edge $e_t$ ending in $t$ such that $\text{lab}(e_1 \cdots e_t) \in L$, we return $e_1 \cdots e_t$. If $e_t$ ends in $t$, but $\text{lab}(e_1 \cdots e_t) \notin L$, we continue with the procedure.
The so-constructed path \( e_1 \cdots e_\ell \) is a solution by construction. Furthermore, since the returned trail has length at most \( |E| \) and there are at most \( |E| \) candidates for the \( i \)th edge, the running time is bounded by a polynomial in \( O(|E|^2 \cdot x) \).

This implies that we can not only find, but also return simple paths and trails that match languages in the tractable classes \( \text{USP}_{\text{tract}} \) and \( \text{UT}_{\text{tract}} \). Since both classes are closed under taking derivatives, see Theorem 3.1, the algorithms \( A_1 \) and \( A_2 \) required in Lemmas 12.3 and 12.4 exist for these languages. This implies the next theorem.

**Theorem 12.6.** The following problems are in polynomial delay:

- \( \text{EnumUSimPaths}(L) \) for each \( L \in \text{USP}_{\text{tract}} \), and
- \( \text{EnumUTrails}(L) \) for each \( L \in \text{UT}_{\text{tract}} \).

### 13. Conclusions and Open Problems

We studied the data complexity of trail and simple path evaluation of RPQs on undirected multigraphs. Although this sounds like a single problem, it is actually a very general class of problems, which subsumes several well-studied problems, such as disjoint-path problems and trail or simple path problems with length constraints.

Using a wide range of methods, such as the minor theorem from Robertson and Seymour [RS95], the minor theorem on group-labeled graphs [Huy09], the extended line graph [KK16], Edmond's matching technique and extensions thereof [Sze03, ADF+08], and several structural arguments, we were able to pinpoint several interesting tractable cases of the problem. Our technically most complex contribution in terms of tractability is the structural argument in the proof of Theorem 7.3.

On the intractable side, we provided the gadget \( G_{3\text{SAT}} \) that can be used to show \( \text{NP}\)-hardness of a wide range of trail and simple path problems (e.g., \( \text{UTrail}((abc)^*) \)) and used directed two-disjoint paths techniques for languages such as \((abab)^*\).

We also compared the tractable classes on undirected (multi-)graphs to the tractable classes on directed (multi-)graphs. We can show that \( \text{SP}_{\text{tract}} \subseteq \text{USP}_{\text{tract}} \) and \( \text{SP}_{\text{tract}} \subseteq \text{UT}_{\text{tract}} \), that is, the languages in \( \text{SP}_{\text{tract}} \) are still tractable under simple path and trail semantics on undirected or bidirectional graphs and multigraphs. On the other hand, \( \text{T}_{\text{tract}} \not\subseteq \text{UT}_{\text{tract}} \), thus \( \text{SP}_{\text{tract}} \) seems more “robust” than \( \text{T}_{\text{tract}} \).

Several challenging open problems remain. The most prominent one is probably the question if it is decidable in polynomial time whether an undirected graph has a simple path of length 0 modulo 3 between two given nodes. We believe that resolving this question will be the key to also resolving the question for larger modulo values. Furthermore, our results seem to suggest that \( \text{USP}_{\text{tract}} \not\subseteq \text{UT}_{\text{tract}} \), but we do not have a proof of the inclusion.

While we did study multigraphs, for all languages for which we showed that \( \text{UTrail} \) or \( \text{USimPath} \) are intractable, we only needed a graph. Since the hardness proofs also imply hardness for multigraphs, the question arises if the class \( \text{UT}_{\text{tract}} \) would be “the same” if we consider edge-labeled graphs or multigraphs. (For \( \text{USP}_{\text{tract}} \), this holds by definition.)

Another interesting direction (which is interesting for designing a query language) is to look for polynomial-time algorithms for two disjoint paths such that the sum of their lengths is minimal.
Acknowledgments. We want to thank Angela Bonifati and Guillaume Bagan for inspiring us to Theorem 8.1, and Tony Huynh for answering questions regarding the simple path of length multiple of 3 problem. This work was funded by the Deutsche Forschungsgemeinschaft (DFG), grant MA 4938/4-1

References


[SBV+21] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid G. Aref, Marcelo Arenas, Maciej Besta, Peter A. Bonez, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Hashofer, Tim Hegeman,


Appendix A. On the Complexity of Properly Edge Colored Disjoint Paths

The work on Gourvès et al. [GdLMM12] on edge-colored graphs contains some flawed proofs, which we discuss here. The flawed proofs are Theorem 9 and Corollary 10. We first repeat their statements in the notation of this thesis, sketch how their proof is flawed, and give an idea how to fix it.

Theorem A.1 (Theorem 9 in [GdLMM12]). Let $G^c$ be a $c$-edge-colored graph with no (almost) PEC closed trails through $s$ or $t$, and a constant $L > 0$. Then, the problem of finding 2 vertex/edge disjoint PEC $s$-$t$ paths, each having at most $L$ edges in $G^c$ is NP-complete in the strong sense, even for graphs with maximum vertex degree equal to 3.

We now restate their theorem in the notion of our paper and neglect some restrictions posed to the graph. Please note that their length $L$ depends on the 3SAT instance, thus it is no “constant” in this sense of this thesis.

Theorem A.2 (restated version of Theorem 9 in [GdLMM12]). Let $G$ be an edge-labeled, undirected graph, $s, t$ two nodes, and $L \in \mathbb{N}$. Then, the problem of finding 2 node-/edge-disjoint $s$-$t$-paths, labeled $(ab)^*$ and both having at most length $L$, is NP-complete.

In their proof, Gourvès et al. consider clause and variable gadgets, similar to the ones presented in Figure 4. (They use a slightly more elaborate clause gadget to achieve node degree at most 3, but we will not go into details here.) Due to the length constraints, they can force a path through the variable gadgets. Yet, they do not see that the path through the clause gadgets still has the opportunity to skip gadgets, using the unused side of a variable gadget. We sketch this in Figure 11. We note that the length constraints given in the original paper allow this skip, as the resulting path will only get shorter.

We note that Theorem A.2 is still correct: From Aboueliam et al. [ADF+08, Theorem 3.2] it follows that two disjoint $(ab)^*/(ab)^*$-paths, one from $s_1$ to $t_1$ and one from $s_2$ to $t_2$ is NP-complete. With length constraints, we can add new nodes $s$ and $t$ and $(ab)^*$-paths of length $L' = 2|E|$ from $s$ to $s_1$ and from $t_2$ to $t$, and $(ab)$-paths of length 2 from $s$ to $s_2$ and from $t_1$ to $t$. Then every path from $s$ to $t$ of length at most $3|E| + 2$ is either from $s_1$ to $t_1$ or from $s_2$ to $t_2$.

For completeness, we note that Theorem A.1 is also correct, which means that the problem is still NP-hard when restricted to graphs without an cycle labeled $(ab)^*$ through $s$ or $t$ (and without an “almost $(ab)^*$ cycle” through $s$ or $t$, which means that every cycle through $s$ or $t$ contains the substring $aa$ or $bb$ at least twice) and with node-degree at most 3. We can prove it with slight changes to $G_{3SAT}$. More precisely, we need to change the clause gadgets similar to [GdLMM12] to ensure that every node has degree at most 3. Furthermore, we add new nodes $s$ and $t$ and use the path length to ensure that every path labeled $(ab)^*$ from $s$ to $t$ which uses the path from $s$ to $s_1$ must not use the $w_r$-labeled paths. This then implies that the path from $s_1$ to $t$ must be via $t_1$ (not $t_2$).

We use $G_{3SAT}$ with the clause gadget depicted in Figure 12 and the words $w_b = a$, $w_m = w_0 = b$, $w_r = a(ab)^i$. We explain $i$ later. We then add new nodes $s$ and $t$ with an edge labeled $(ab)^j$ from $s$ to $s_1$ and edges labeled $ab$ from $s$ to $s_2$, and labeled $b$ from $t_1$ to $t$ and $t_2$ to $t$. We choose $i$ and $j$ such that $i + j > L$, while the intended path from $s_2$ to $t_2$ does not exceed length $L$. Let $m$ be the number of clauses of $\varphi$ and $n$ be the number of variables. The length of an “intended” path from $s_1$ to $t_1$ is $3m(2|w_b| + 2|w_0|) - |w_0| = 12m - 1$, therefore the length of an indented path from $s$ to $t$ via $s_1$ is $12m + j$. The length of an “intended” path from $s_2$ to $t_2$ is $m(4|w_r| + 4|w_0|) + n(3|w_r| + 3|w_0|) - |w_0| = m(8i + 8) + n(6i + 6) - 1.$
THE COMPLEXITY OF RPQS ON UNDIRECTED GRAPHS

We set \( i = 12m + 1, \ L = 96m^2 + 16m + 72n + 12n + 2 \) and \( j = L - 12m \) to complete the construction.

The length constraints imply that the path from \( s \) to \( t \) that uses the path from \( s \) to \( s_1 \) must not use \( w_r \)-paths. By construction, this path must then enter \( t \) via the path from \( t_1 \). Thus we have a path from \( s_1 \) to \( t_1 \) which must not use any of the \( w_r \)-paths. The correctness then follows similar to the proof of Theorem 4.1.

**Corollary A.3** (Corollary 10 in [GdLMM12]). Let \( s \) and \( t \) be two vertices in a \( c \)-edge-colored graph \( G^c \) with maximum vertex degree equal to 3 and with no (almost) PEC closed trails through \( s \) or \( t \). Then, it is NP-complete to decide whether there exist 2 vertex/edge disjoint \( s \)-\( t \) paths such that exactly one of them is a PEC \( s \)-\( t \) path.

**Corollary A.4** (restated version of Corollary 10 in [GdLMM12]). Let a graph \( G \) and nodes \( s, t \) be given. Then it is NP-complete to decide whether there exist 2 node-/edge-disjoint \( s \)-\( t \)-paths such that one of them matches \((ab)^*\).
In their proof, Gourvès et al. use the same construction as in their Theorem 10, but color the clause gadgets blue, that is, all edges in the clause gadget get the label \( a \). Since one path has to be PEC, that is, match \((ab)^*\), this again forces one of the paths to go through each variable gadget. But, similar to before, the path using the clause gadgets can shortcut via the unused side of variable gadgets.

We note that Corollary A.4 and Corollary A.3 are correct. Indeed, we can use the variant of \( G_{3\text{SAT}} \) which we used to prove Theorem A.1 and choose \( w_r = aa \) and relabel the path from \( s \) to \( s_2 \) with \( aa \). Then the path from \( s \) to \( t \) that matches \((ab)^*\) must use the \((ab)^*\)-labeled path from \( s \) to \( s_1 \) and must not use \( w_r \)-paths. This implies that the path must enter \( t \) via \( t_1 \). Since the path from \( s_1 \) to \( t_1 \) must not use \( w_r \)-paths, and the other path from \( s \) to \( t \) must be node-/or edge-disjoint and therefore use the disjoint path from \( s_2 \) to \( t_2 \), the correctness follows from Theorem 4.1.

University of Bayreuth, Germany