A Trichotomy for Regular Simple Path Queries on Graphs

Guillaume Bagan
INRIA, France
guillaume.bagan@inria.fr

Angela Bonifati
Lille 1 University & INRIA, France
angela.bonifati@inria.fr

Benoit Groz
Tel-Aviv University, Israel
benoit.groz@crans.org

ABSTRACT

Regular path queries (RPQs) select vertices connected by some path in a graph. The edge labels of such a path have to form a word that matches a given regular expression. We investigate the evaluation of RPQs with an additional constraint that prevents multiple traversals of the same vertices. Those regular simple path queries (RSPQs) quickly become intractable, even for basic languages such as $(aa)^*$ or a^*ba^*.

In this paper, we establish a comprehensive classification of regular languages with respect to the complexity of the corresponding regular simple path query problem. More precisely, we identify for which languages RSPQs can be evaluated in polynomial time, and show that evaluation is NP-complete for languages outside this fragment. We thus fully characterize the frontier between tractability and intractability for RSPQs, and we refine our results to show the following trichotomy: evaluation of RSPQs is either AC^0, NL-complete or NP-complete in data complexity, depending on the language L. The fragment identified also admits a simple characterization in terms of regular expressions.

Finally, we also discuss the complexity of deciding whether a language L belongs to the fragment above. We consider several alternative representations of L: DFAs, NFAs or regular expressions, and prove that this problem is NL-complete for the first representation and PSPACE-complete for the other two. As a conclusion we extend our results from edge-labeled graphs to vertex-labeled graphs.

Categories and Subject Descriptors: G.2.2, F.2.0

General Terms: Algorithms, Languages, Theory

Keywords: Graphs, Paths, Regular Simple Paths, Complexity, Regular Languages, Automata.

1. INTRODUCTION

The reachability problem for graphs (finding a path between two vertices) has been heavily investigated in computer science, and admits very efficient algorithms. However, for many real-world problems, constraints on the path need to be considered and, as a consequence, the reachability problem can become computationally hard. Constrained path problems on regular paths are of particular interest. For graph databases, such problems have been examined in the context of regular path queries (RPQs). Given a language L and two vertices in a database graph, a regular path query selects pairs of vertices connected by a path whose edge labels form a word in L. Graph databases and RPQs have been investigated starting from the late 80s [1, 6, 9, 10, 11, 14, 15, 19, 28, 30], and are now again in vogue due to their wide application scenarios, e.g. in social networks [36], biological and scientific databases [27, 33], and the Semantic Web [18]. Regular path queries allow to traverse the same vertices multiple times, whereas regular simple path queries (RSPQs) permit to traverse each vertex only once. From a theoretical viewpoint, the former notion has overridden the latter, mainly for complexity reasons. Indeed, RPQs are computable in time polynomial in both query and data complexity (combined complexity), while the evaluation of RSPQs is NP-complete even for fixed basic languages such as $(aa)^*$ or a^*ba^* [30]. RSPQs, however, are desired in many application scenarios [27, 33, 7, 25, 23, 41], such as transportation problems, VLSI design, metabolic networks, DNA matching and routing in wireless networks. As a further example, the problem of finding subgraphs matching a graph pattern can be generalized to use regular expressions on pattern edges [15]. Such queries may also enforce the condition that their matched vertices are distinct. Additionally, regular simple paths have been recently considered in SPARQL 1.1 queries exhibiting property paths. In particular, recent studies on the complexity of property paths in SPARQL [3, 29] have highlighted the hardness of the semantics proposed by W3C to evaluate such paths in RDF graphs. Roughly speaking, according to the semantics considered in [29], the evaluation of expressions under Kleene-star closure imposes that the involved path is simple, whereas the evaluation of the remaining expressions allows to traverse the same vertex multiple times. As such, the semantics studied in [29] is an hybrid between regular paths and regular simple paths semantics.

Contributions. In this paper, we address the long standing open question [30, 7] of exactly characterizing the maximal class of regular languages for which RSPQs are tractable. By "tractable" we mean computable in time polynomial in the size of the database. Precisely, we establish a comprehensive classification of the complexity of RSPQs for a fixed regular language L: given an edge-labeled graph G and two vertices x and y, is there a simple path from x to y whose edge labels

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS'13, June 22–27, 2013, New York, New York, USA.
Copyright 2013 ACM 978-1-4503-2066-5/13/06 ...$15.00.

form a word of L? A first step towards this important issue has been made in [30]. They exhibit a tractable fragment: the class of languages closed by subword. However, their fragment is not maximal.

Our contributions can be detailed as follows. We introduce a class of languages, named $\mathcal{C}_{tract}$, for which RSPQs are computable in polynomial time, and even in NL. We then show that this fragment is maximal as the RSPQ problem is NP-complete for every regular language that does not belong to $\mathcal{C}_{tract}$. Consequently, we characterize the frontier between tractability and intractability for this problem, under the hypothesis NL $\neq$ NP that is actually weaker than PTIME $\neq$ NP. Additionally, $\mathcal{C}_{tract}$ also represents the maximal class for which finding a shortest path that satisfies a RSPQ is tractable. We note that we focus on data complexity as we assume that the language L is fixed. At this point, the chart of the classification of the languages is not yet complete. Therefore, we refine our results to show the following trichotomy: the RSPQ problem is either AC^0, NL-complete or NP-complete.

We discuss the complexity to decide, given a language L, whether the RSPQ problem for L is tractable. We consider several alternative representations of L: DFAs, NFAs or regular expressions. We prove that this problem is NL-complete for the first representation and PSPACE-complete for the two others.

Next, we give a characterization of the tractable fragment $\mathcal{C}_{tract}$ for edge-labeled graphs in term of regular expressions. Moreover, $\mathcal{C}_{tract}$ is closed by union and intersection and languages in $\mathcal{C}_{tract}$ are aperiodic i.e. can be expressed by first-order formulas [39].

The above results hold for the usual definition of database graphs, i.e. edge-labeled graphs. However, it seems natural to take into consideration both queries on top of vertices labels and queries on top of vertices and edges labels. As an example, a Google Maps user may be interested to specify as a condition a regular expression that enforces a stop over in a given city and avoids another city while preferring certain types of roads. For such a reason, we focus on another model: vertex-labeled graphs. Surprisingly, for some languages, the RSPQ problem is simpler on vertex-labeled graphs than on edge-labeled graphs. With $L = (ab)^*$ for instance, RSPQ is polynomial for vertex-labeled graphs and NP-complete for edge-labeled graphs. Furthermore, we can adapt our results to prove, for this model, a classification of the same kind as the one shown for vertex-labeled graphs: the RSPQ problem is either AC^0, NL-complete or NP-complete.

As a final contribution, we have obtained two minor results. First, we have attempted to study the parametrized complexity of tractable RSPQs queries when the parameter is the size of the query. However, we obtained a partial result: we prove that the problem is FPT for the class of finite languages. Moreover, we prove that the problem is also FPT for the class of all regular languages when the parameter is the size of the path. As a second result, we prove that the problem RSPQ is polynomial w.r.t. combined complexity on graphs of bounded directed treewidth. This is actually a straightforward generalization of a result of [24].

Related Work. Regular path queries express ways to evaluate regular expressions on database graph models [1, 6, 9, 10, 11, 14, 15, 19, 28, 30] or tree-structured models, such as XML [12]. While the regular path problem has been extensively studied in the literature, the regular simple path problem has received less attention in both the database and graph communities. Besides the works on regular paths, there have been studies on finding paths with some constraints. In particular, Lapaugh et al. [26] prove that finding simple paths of even length is polynomial for non directed graphs and NP-complete for directed graphs. This study has been extended in [4] by considering paths of length i mod k. Similarly, finding k disjoint paths with extremities given as input is polynomial for non directed graphs [35] and NP-complete for directed graphs [17]. Mendelzon and Wood [30] show that the regular simple path problem is NP-complete in the general case. However, they show that the problem can be decided in polynomial time for subword-closed languages. They also show that the problem becomes polynomial under some restrictions on the size of cycles of both graph and automaton. A subsequent paper [32] proves the polynomiality for the class of outerplanar graphs. Barrett et al. [7] extend this result, proving that the regular simple path problem is polynomial w.r.t. combined complexity for graphs of bounded treewidth. Let us also observe that the existence of a regular simple path between two vertices is MSO-definable, and therefore a well-known result of Courcelle [13] already implies the same result but w.r.t. data complexity only. Barrett et al. [7] also show that the problem is NP-complete for the class of grid graphs even when the language is fixed. Practical algorithms for regular simple paths on large graphs have been proposed in [23, 25].

Regular simple paths have been also investigated in the context of SPARQL property paths with the semantics proposed in a working draft of SPARQL 1.1. Notice that such semantics of SPARQL property paths doesn't exactly correspond to regular simple paths queries. Losemann and Martens [29] and Arenas et al. [3] investigate the complexity of evaluating such property paths. They show that the evaluation is NP-complete in several cases, along with exhibiting cases in which it is polynomial. More precisely, Losemann and Martens consider different fragments of regular expressions and classify them with respect to the complexity of evaluating SPARQL property paths. Both papers also show that counting the number of paths that match a regular expression (which is permitted by the working draft) is hard in many cases.

2. PRELIMINARIES

For the rest of the paper, Σ always denotes a finite alphabet. We use the notation $[n]$ to denote the set of integers $\{1, \ldots, n\}$. Given a word w and a language L, $w^{-1}L = \{w' : ww' \in L\}$.

Complexity: NL, P, NP, PSPACE refer to the classical classes of complexity [34]. The reductions we consider are many-to-one logspace reductions [34] and completeness of problems are under this type of reductions. For some proofs, we consider non deterministic algorithms (Turing machines) using oracles. We use the Ruzzo-Simon-Tompa model [37]. In this model, the Turing machine needs to be deterministic while it makes queries to the oracle. The class AC^0 refers to uniform AC^0 that is equivalent to $FO(+, \times)$ or $FO(BIT, <)$ [22]. For definition of FPT, see [16].

Graphs: In our paper, we essentially consider db-graphs even if we also consider vertex-labeled graphs. A db-graph is a tuple $G = (V, \Sigma, E)$ where V is a set of vertices, Σ is a set of labels and $E \subseteq V \times \Sigma \times V$ is a set of edges labeled

by symbols of Σ. A path p of a db-graph G from x to y is a sequence $(v_1 = x, a_1, \ldots, v_m, a_m, v_{m+1} = y)$ such that for each $i \in [m+1]$, v_i is a vertex in G and for each $i \in [m]$, (v_i, a_i, v_{i+1}) is an edge in G. A path p is simple if all vertices v_i in p are distinct. Given a language $L \subseteq \Sigma^*$, p is L-labeled if $a_1 \ldots a_m \in L$. Given a subset $S \subseteq V$, p is S-restricted if every intermediate vertex of p belongs to S. Given a simple path p and two vertices x and y in p, $p[x, y]$ denotes the subpath of p from x to y.

Languages and automata: Let L be a regular language. We denote by $A_L = (Q_L, i_L, F_L, \Delta_L)$ the minimal DFA for L, and by M_L the number of states $M_L = |Q_L|$ in A_L. Whenever the language is obvious from context, we drop the subscript and write M instead of M_L. We assume that A_L is complete i.e. Δ_L is a total function, so that in general A_L may have a sink state. For any $q \in Q, w \in \Sigma^*$, $\Delta_L(q, w)$ denotes the state obtained when reading w from q. Finally, $\mathscr{L}_q$ denotes the set of all words accepted from q. For every state q we denote by $Loop(q)$ the set of all non empty words that allow to loop on q: $Loop(q) = \{w \in \Sigma^+ \mid \Delta_L(q, w) = q\}$. Strongly connected components of (the graph of) A_L are simply called components. The run of L (or A_L) over a p be a path $p = (v_1, a_1, \ldots, a_m, v_{m+1})$ is the mapping $\rho : \{v_1, \ldots, v_{m+1}\} \mapsto Q_L$ such that: $\rho(v_1) = i_L$ and $\rho(v_{i+1}) = \Delta_L(i_L, a_1 \ldots a_i)$ for every $i \in [m]$. There are many characterisations of aperiodic languages [39]. A language L is aperiodic if and only if it satisfies $\Delta_L(q, w^{M+1}) = \Delta_L(q, w^M)$ for every state q and word w.

RSPQ: Given a class $\mathcal{L}$ of regular languages and a class $\mathcal{G}$ of db-graphs, we define the following problem:

RSPQ$(\mathcal{L}, \mathcal{G})$
Input: a language $L \in \mathcal{L}$,
a db-graph $G = (V, \Sigma, E) \in \mathcal{G}$,
and two vertices $x, y \in V$
Question: is there a simple L-labeled path from x to y?

The encoding of the language L will be specified when required. We denote by "All" the class of db-graphs, RSPQ$(\mathcal{L})$ means RSPQ$(\mathcal{L}, \text{All})$. For any single language L, we use RSPQ$(L, \mathcal{G})$ to denote RSPQ$(\{L\}, \mathcal{G})$. Since L is fixed, we focus on data complexity. Notice that the representation of L does not matter here. Although we consider the boolean version of the problem, namely deciding the existence of a path, our algorithms actually also return a simple L-labeled path.

Given a regular language L, our main question is to give a criterion to decide whether RSPQ(L) is tractable (i.e. decidable in polynomial time) or not (i.e. NP-complete). We address this question in the next and following sections.

Example 1. *As an introductory example, consider the language $L = a^*(bb^+ + \epsilon)c^*$. We wish to decide whether there exists a simple path from x to y labeled by L, given two vertices x, y of a db-graph G. It is not absolutely trivial that this problem can be solved efficiently: the problem has indeed been proved* NP*-complete for the language a^*bc^*. Yet we can give a polynomial algorithm for L.*

We distinguish two cases: there is a simple L-labeled path from x to y if and only if one of the following cases holds:

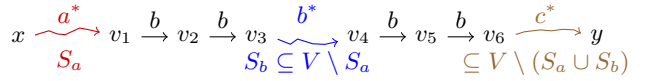
1. *there exists a simple $a^*b^kc^*$-labeled path from x to y for some $k \in \{0, 2, 3\}$*
2. *case 1 does not hold and there exists a simple $a^*b^4b^*c^*$-labeled path from x to y.*

The first case is the easiest to check. We essentially enumerate all possible combinations for the k b-labeled edges, for increasing values of k. For each such combination one only needs to check the following conditions: (1) there is an a^*-labeled path leading from x to the first b-edge (without crossing the other b-edges) (2) there is a c^*-labeled path leading from the last b-edge to y (without crossing the other b-edges). For details about why these verifications suffice to check case 1, we refer the reader to the long version [5].

Let us now assume w.l.o.g. that there is no $a^*b^kc^*$-labeled path from x to y for $k \in \{0, 2, 3\}$. We can show that in this second case there exists a simple L-labeled path from x to y if and only if there exist six vertices $v_1, v_2, v_3, v_4, v_5, v_6$, all distinct except that v_3 may equal v_4, two integers l_a, l_b and two sets S_a, S_b satisfying all following conditions:

- *there is a b-labeled edge from v_1 to v_2, from v_2 to v_3, from v_4 to v_5, and from v_5 to v_6.*
- *there is an a^*-labeled path from x to v_1 avoiding all other v_is $(i > 1)$. The shortest possible such path has length l_a.*
- *S_a is the set of all vertices reachable from x through an a^*-labeled path of length at most l_a that avoids all other v_is $(i > 1)$.*
- *there is a b^*-labeled path from v_3 to v_4 of which all vertices (but the first and last) avoid S_a and the v_is. The shortest possible such path has length l_b.*
- *S_b is the set of all vertices reachable from v_3 through any b^*-labeled path of length at most l_b that avoids S_a and all other v_is $(i \neq 4)$.*
- *a c^*-labeled path from v_6 to y of which all vertices (but the first) avoid S_a and S_b and all other v_is $(i < 6)$.*

The figure below summarizes all these conditions.



These conditions can clearly be verified in time polynomial in G. It is relatively clear also that the path constructed above is a simple L-labeled path from x to y, so the conditions are sufficient to obtain an L-labeled simple path. Proving that they are necessary requires a little more attention: why should we indeed avoid all vertices from S_a when building the path from v_3 to v_4? The reason why we choose to avoid a superset of the subpath from x to v_1 instead of the subpath itself is obvious: we want our algorithm to be "memoryless" in the sense that the a^* and b^* subpaths should be constructed independently, lest we enumerate exponentially many paths. But how can we ascertain that we do not overlook a simple L-labeled path when do not find any? For the present, we only claim that every shortest L-labeled simple path from x to y yields vertices $v_1, \ldots, v_6$ satisfying the conditions above. A sketch of proof is given in the long version for this specific example, and we develop in this paper the general idea underlying this argument to elaborate an algorithm for tractable instances.

3. A TRICHOTOMY FOR RSPQ

Here and henceforth, M refers to the size of Q_L. We next define a class of languages. We will prove that it is exactly the class of regular languages for which RSPQ(L) is tractable.

DEFINITION 1. *We define $\mathcal{C}_{tract}$ as the class of regular languages L that satisfy $w^M \mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$ for all pairs of states $q_1, q_2 \in Q_L$ and word w such that $Loop(q_1) \neq \emptyset$, $Loop(q_2) \neq \emptyset$, $q_2 \in \Delta_L(q_1, \Sigma^*)$ and $w \in Loop(q_2)$.*

This definition is merely a technical definition for the $\mathcal{C}_{tract}$, but Theorem 4 provides more intuitive characterizations of the class.

3.1 Hard languages for RSPQ

This section is devoted to the proof of a hardness result: RSPQ(L) is NP-hard for every regular language L that does not belong to $\mathcal{C}_{tract}$.

The first step toward that proof lies in the following characterization of $\mathcal{C}_{tract}$. We call *witness for hardness* a tuple (q, w_m, w_r, w_1, w_2) where $q \in Q_L$, $w_r \in \Sigma^*$, $w_1 \in Loop(q)$ and $w_2, w_m \in \Sigma^+$ satisfying

- $w_m w_2^* w_r \subseteq \mathscr{L}_q$
- $(w_1 + w_2)^* w_r \cap \mathscr{L}_q = \emptyset$.

LEMMA 1. *Let L be a regular language that does not belong to $\mathcal{C}_{tract}$. Then, L admits a witness for hardness.*

PROOF. Let L be a regular language that does not belong to $\mathcal{C}_{tract}$. For commodity, we distinguish two cases, depending on whether L satisfies or not the following property: $\mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$ for every $q_1, q_2 \in Q_L$ such that $q_2 \in \Delta_L(q_1, \Sigma^*)$ and $Loop(q_1) \cap Loop(q_2) \neq \emptyset$ (Property $\mathcal{P}$) .

We first prove that every language satisfying property $\mathcal{P}$ is aperiodic. Let L be a language satisfying property $\mathcal{P}$, $q \in Q_L$ and w a word. Let also q' denote the state $q' = \Delta_L(q, w^M)$. We denote by q'' the state $\Delta_L(q', w)$. We want to prove that $q' = q''$. It is trivial when $w = \epsilon$. Thus, we assume that $w \neq \epsilon$. According to the pumping lemma, there exists some k such that $\Delta_L(q', w^k) = q'$. Then q' and q'' both loop on w^k, so that $\mathscr{L}_{q'} = \mathscr{L}_{q''}$ by definition of $\mathcal{P}$, hence $q' = q''$ by minimality. Consequently, L is aperiodic.

If L does not satisfy Property $\mathcal{P}$, there exist q, q_2, w_m, w, w_r such that $\Delta_L(q, w_m) = q_2$, $w \in Loop(q) \cap Loop(q_2)$, and $w_r \in \mathscr{L}_{q_2} \setminus \mathscr{L}_q$. Then $q, w_m, w_r, w_1 = w_2 = w$ is a witness for hardness.

Assume now that L still does not belong to $\mathcal{C}_{tract}$, but satisfies Property $\mathcal{P}$, and so in particular is aperiodic. By definition of $\mathcal{C}_{tract}$ there exist states q, q_2 and words w_1, w_2, w_m, w'_r such that $w_1 \in Loop(q)$, $w_2 \in Loop(q_2)$, $\Delta_L(q, w_m) = q_2$, $w'_r \in \mathscr{L}_{q_2}$ and $w_2^M w'_r \notin \mathscr{L}_q$. W.l.o.g. we can suppose that $w_1 = (w'_1)^M$ for some word w'_1. We then claim that $\mathscr{L}_{q'} \subseteq \mathscr{L}_q$ for every q' in $\Delta_L(q, \Sigma^* w_1)$. Indeed, every $q' \in \Delta_L(q, \Sigma^* w_1)$ loops over w_1 by the pumping lemma and aperiodicity of L, hence $w_1 \in Loop(q) \cap Loop(q')$ and therefore $\mathscr{L}_{q'} \subseteq \mathscr{L}_q$ due to Property $\mathcal{P}$.

Let $w_r = w_2^M w'_r$. By definition, $w_m w_2^* w_r \subseteq \mathscr{L}_q$ because $w_r \in \mathscr{L}_{q_2}$. We now prove that $(w_1 + w_2)^* w_r \cap \mathscr{L}_q = \emptyset$, because any word in $(w_1 + w_2)^* w_r$ can be decomposed into uv with $u \in \epsilon + (w_1 + w_2)^* w_1$ and $v \in (w_2)^* w_r$. We recall that $w_r = w_2^M w'_r \notin \mathscr{L}_q$ and L is aperiodic, so that $v \notin \mathscr{L}_q$. Furthermore, we have just proved that $q' = \Delta_L(q, u)$ satisfies $\mathscr{L}_{q'} \subseteq \mathscr{L}_q$. Consequently, $v \notin \mathscr{L}_{q'}$ and $uv \notin \mathscr{L}_q$. Thus, q, w_m, w_r, w_1, and w_2 provide a witness for hardness, which concludes the proof of Lemma 1. □

We can now prove our hardness result, by reduction from Vertex-Disjoint-Path, a problem also used in [30] to prove hardness in the particular case of a^*ba^*.

Vertex-Disjoint-Path
Input: A directed graph $G = (V, E)$, four vertices $x_1, y_1, x_2, y_2 \in V$
Question: are there two disjoint paths, one from x_1 to y_1 and the other from x_2 to y_2?

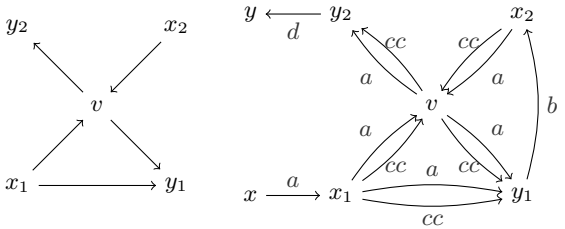
LEMMA 2. *Let L be a regular language that does not belong to $\mathcal{C}_{tract}$. Then, RSPQ(L) is* NP*-hard.*

PROOF. Let $L \notin \mathcal{C}_{tract}$. We exhibit a reduction from the Vertex-Disjoint-Path problem to RSPQ(L). According to Lemma 1, L admits a witness for hardness q, w_m, w_r, w_1, w_2. Let w_l be a word such that $\Delta(i_L, w_l) = q$. By definition we get $w_l(w_1 + w_2)^* w_r \cap L = \emptyset$ and $w_l w_1^* w_m w_2^* w_r \subseteq L$.

We build from G a db-graph G' whose edges are labeled by non empty words. This is actually a generalization of db-graphs. Nevertheless, by adding intermediate vertices, an edge labeled by a word w can be replaced with a path whose edges form the word w.

G' is constructed as follows. The vertices of G' are the same as the vertices of G. For each edge (v_1, v_2) in G, we add two edges (v_1, w_1, v_2) and (v_1, w_2, v_2). Moreover, we add two new vertices x, y and three edges (x, w_l, x_1), (y_1, w_m, x_2) and (y_2, w_r, y).

By construction, for every simple path p from x to y in G' that contains the edge (y_1, w_m, x_2), we can obtain a similar path that matches a word in $w_l w_1^* w_m w_2^* w_r$ by switching w_1 and w_2 edges, keeping the same vertices. Every simple path p from x to y in G' that does not contain the edge (y_1, w_m, x_2) matches a word in $w_l(w_1 + w_2)^* w_r$. By definition of q, w_m, w_r, w_1, w_2, no path of that form $w_l(w_1 + w_2)^* w_r$ is L-labeled, whereas every path matching $w_l w_1^* w_m w_2^* w_r$ is L-labeled. Thus, RSPQ(L) returns True for (G', x, y) iff there is a simple path from x to y in G' that contains the edge (y_1, w_m, x_2) that is, iff Vertex-Disjoint-Path returns True for (G, x_1, y_1, x_2, y_2). We illustrate below the reduction for $L = a^*b(cc)^*d$, on an instance (G, x_1, y_1, x_2, y_2), choosing $w_l = w_1 = a$, $w_m = b$, $w_2 = cc$, and $w_r = d$. □



Input instance G RSPQ instance: graph G'

Figure 1: Reduction for $L = a^*b(cc)^*d$.

3.2 Tractable languages for RSPQ

The main result of this section is that for every $L \in \mathcal{C}_{tract}$, RSPQ($L$) $\in$ NL. The general idea is to exploit a particular kind of pumping argument between strongly connected components of the automaton to prove that if we build carefully a path using the usual reachability algorithm inside the strongly connected components, then we need not care about possible intersections between subpaths corresponding to different components. For this purpose, we first prove several lemmas on the structure of automata that recognize $\mathcal{C}_{tract}$ languages. To begin with, we prove that every language from $\mathcal{C}_{tract}$ is aperiodic and deduce an alternative characterization of $\mathcal{C}_{tract}$.

3.2.1 Alternative characterization of $\mathcal{C}_{tract}$.

LEMMA 3. *Let L be a regular language in $\mathcal{C}_{tract}$. Then L is aperiodic.*

PROOF. In the proof of Lemma 1 we defined a property $\mathcal{P}$ and showed that languages satisfying property $\mathcal{P}$ are aperiodic. We show that every $L \in \mathcal{C}_{tract}$ satisfies property $\mathcal{P}$. Let $L \in \mathcal{C}_{tract}$, $q_1, q_2 \in Q_L$ and w satisfy $q_2 \in \Delta_L(q_1, \Sigma^*)$ and $w \in Loop(q_1) \cap Loop(q_2)$. By definition of $\mathcal{C}_{tract}$, $w^M \mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$, hence $\mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$ because $w \in Loop(q_1)$. □

We then exploit this aperiodicity property to establish the following characterization of $\mathcal{C}_{tract}$, which strengthens the requirements from Definition 1 on the loops of A_L.

LEMMA 4. *Let L be a regular language. Then, L belongs to $\mathcal{C}_{tract}$ iff for all pair of states $q_1, q_2 \in Q_L$ such that $Loop(q_1) \neq \emptyset$, $Loop(q_2) \neq \emptyset$ and $q_2 \in \Delta_L(q_1, \Sigma^*)$, the following statement holds: $(Loop(q_2))^M \mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$.*

PROOF. The (if) implication is immediate by Definition 1. Let us now prove the (only if) implication. Assume $L \in \mathcal{C}_{tract}$. Let $q_1', q_2' \in Q_L$ satisfy $Loop(q_1') \neq \emptyset$, $Loop(q_2') \neq \emptyset$, $q_2' \in \Delta_L(q_1', \Sigma^*)$, and let $w \in Loop(q_2')$. Let also q_3 denote the state $\Delta_L(q_1', w^M)$. Then Definition 1 implies $w^M \mathscr{L}_{q_2'} \subseteq \mathscr{L}_{q_1'}$. Thus, $\mathscr{L}_{q_2'} \subseteq \mathscr{L}_{q_3}$. The crux of the proof is to choose carefully q_1', q_2' and w to exploit the constraints on $\mathscr{L}_{q_3}$.

Let q_1, q_2 be two states such that $Loop(q_1) \neq \emptyset$, $Loop(q_2) \neq \emptyset$ and $q_2 \in \Delta_L(q_1, \Sigma^*)$. Let $v_1 \dots v_M \in (Loop(q_2))^M$ and $q_3 = \Delta_L(q_1, v_1 \dots v_M)$. We wish to prove $\mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_3}$.

For some i, j, $1 \leq i < j \leq M$, we get $\Delta_L(q_1, v_1 \dots v_i) = \Delta_L(q_1, v_1 \dots v_j)$, using the convention $\Delta_L(q_1, v_1 \dots v_i) = q_1$ for $i = 0$. Let $u_1 = v_1 \dots v_i$, $u_2 = v_{i+1} \dots v_j$ and $u_3 = v_{j+1} \dots v_k$. Let $q_4 = \Delta_L(q_1, u_1)$. We claim that $\mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_4}$. The result then follows from $\mathscr{L}_{q_2} = u_3^{-1} \mathscr{L}_{q_2} \subseteq u_3^{-1} \mathscr{L}_{q_4} = \mathscr{L}_{q_3}$. To prove the claim, let $w = u_1 u_2^M$ and $q_5 = \Delta_L(q_1, w^M)$. As $\Delta_L(q_1, w^M) = q_5$ and $w \in Loop(q_2)$, we get $\mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_5}$ through Definition 1 with q_1, q_2 and w. Furthermore, u_2 belongs to $Loop(q_5)$ because L is aperiodic. To conclude the proof, we observe that $\mathscr{L}_{q_5} \subseteq \mathscr{L}_{q_4}$, by Definition 1 with q_5, q_4 and u_2, and because $\Delta_L(q_4, u_2^M) = q_4$ and $u_2 \in Loop(q_5)$.[1] □

3.2.2 Technical lemmas on the components of A_L

From now on, and until the end of the section, we fix a language $L \in \mathcal{C}_{tract}$. We introduce in Lemmas 7 and 9 the pumping argument that we exploit in the algorithm to compute a simple path. In the other lemmas we prove auxiliary results, based on the decomposition of the automaton in strongly connected components. We prove that components of languages in $\mathcal{C}_{tract}$ are very particular, in the sense that every word staying long enough in the component is synchronizing. A preliminary lemma shows that that two distinct states q_1 and q_2 in the same component cannot loop on the same word.

LEMMA 5. *Let q_1, q_2 two states belonging to the same component of A_L. If $Loop(q_1) \cap Loop(q_2) \neq \emptyset$, then $q_1 = q_2$.*

PROOF. Let q_1, q_2 as above, and let w a word in $Loop(q_1) \cap Loop(q_2)$. According to Definition 1, $w^M \mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$, hence $\mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$ since $w \in Loop(q_1)$. By symmetry, $\mathscr{L}_{q_2} = \mathscr{L}_{q_1}$, which implies $q_2 = q_1$. □

The next two lemmas characterize the internal language of a component.

LEMMA 6. *Let C be a component of A_L, $q_1, q_2 \in C$ and $a \in \Sigma$. Then $\Delta_L(q_1, a) \in C$ iff $\Delta_L(q_2, a) \in C$.*

PROOF. Let $q_1 \neq q_2$ two states in the same component C. Let a satisfy $\Delta_L(q_1, a) \in C$. Let also $w \in Loop(q_1) \cap a\Sigma^*$ and $q_3 = \Delta_L(q_2, w^M)$. We will prove that q_3 belongs to C and consequently $\Delta_L(q_2, a) \in C$. As L is aperiodic, $w \in Loop(q_3)$, and consequently, $w^M \mathscr{L}_{q_3} \subseteq \mathscr{L}_{q_1}$ by Definition 1. Furthermore, $w^M \mathscr{L}_{q_1} \subseteq \mathscr{L}_{q_2}$ also by Definition 1. Hence $\mathscr{L}_{q_3} \subseteq \mathscr{L}_{q_1}$ and $\mathscr{L}_{q_1} \subseteq (w^M)^{-1} \mathscr{L}_{q_2} = \mathscr{L}_{q_3}$. Thus, $\mathscr{L}_{q_1} = \mathscr{L}_{q_3}$ and, by minimality of A_L, $q_1 = q_3$, so that $q_3 \in C$. □

NOTATION 1. *We denote the internal alphabet of a component C of A_L by $\Sigma_C = \{a \in \Sigma : \exists q_1, q_2 \in C. \Delta_L(q_1, a) = q_2\}$.*

As a direct consequence of Lemma 6 we get:

LEMMA 7. *Let C be a component of A_L, $q \in C$ and $w \in \Sigma^*$. Then $\Delta_L(q, w) \in C$ iff $w \in (\Sigma_C)^*$.*

Finally, we prove that inside a component, every word with length at least M^2 is synchronizing. This result is the core of our pumping argument between strongly connected components as exposed in Lemma 9.

LEMMA 8. *Let C be a component of A_L, Σ_C be the internal alphabet of C, q_1, q_2 be two states of C and $w \in (\Sigma_C)^{M^2}$. Then, $\Delta_L(q_1, w) = \Delta_L(q_2, w)$.*

PROOF. Assume that $w = a_1 \dots a_{M^2}$. For each i from 0 to M^2 and $\alpha = 1, 2$, let $q_{\alpha,i} = \Delta_L(q_\alpha, a_1 \dots a_i)$. Since there are at most M^2 distinct pairs $(q_{1,i}, q_{2,i})$, there exist i, j, with $i < j$ such that $q_{1,i} = q_{1,j}$ and $q_{2,i} = q_{2,j}$. By Lemma 7, $q_{1,i}, q_{2,i} \in C$. Let $w' = a_{i+1} \dots a_j$. We have $w' \in Loop(q_{1,i}) \cap Loop(q_{2,i})$, hence $q_{1,i} = q_{2,i}$ by Lemma 5. As a consequence, $\Delta_L(q_1, w) = \Delta_L(q_2, w)$. □

Here and thereafter, we fix the constant $N = 2M^2$.

LEMMA 9. *Let q_1, q_2 be two states such that $Loop(q_1) \neq \emptyset$, $Loop(q_2) \neq \emptyset$, and $q_2 \in \Delta_L(q_1, \Sigma^*)$. Let C be the component that contains q_2 and Σ_C be the internal alphabet of C. Then, $\mathscr{L}_{q_2} \cap (\Sigma_C)^N \Sigma^* \subseteq \mathscr{L}_{q_1}$.*

[1] This last application of Definition 1 corresponds actually to observing that every language in $\mathcal{C}_{tract}$ satisfies property $\mathcal{P}$ from Lemma 1.

PROOF. Let $w \in \mathscr{L}_{q_2} \cap (\Sigma_C)^N \Sigma^*$. There are some words $u, v \in (\Sigma_C)^{M^2}$, $w' \in \Sigma^*$ such that $w = uvw'$. By Lemma 7 and the Pigeonhole Principle, there exist a state $q_3 \in C$ and $M+1$ non-empty words $v_1, \ldots, v_{M+1}$ such that $v = v_1 \ldots v_{M+1}$ and $\Delta_L(q_2, uv_1 \ldots v_i) = q_3$ for every $i \in [M]$. Therefore, $w \in uv_1(Loop(q_3))^{M-1} v_{M+1} w'$. By Lemma 8, $\Delta_L(q_3, uv_1) = \Delta_L(q_2, uv_1) = q_3$. Thus, w belongs to both $(Loop(q_3))^M v_{M+1} w'$ and $\mathscr{L}_{q_3}$. By Lemma 4, $w \in \mathscr{L}_{q_1}$. □

3.2.3 *Computing* RSPQ(L) *for* L *in* $\mathcal{C}_{tract}$

In the following, we describe a polynomial algorithm that computes RSPQ(L) when L belongs to $\mathcal{C}_{tract}$. Observe that a dynamic programming approach can be used to obtain a non necessarily simple regular path with label L between two points. This is because such paths can be built incrementally by storing only the last vertex in the (partial) path together with the corresponding state. This approach is not adequate to build a *simple* path, as we need to memorize all the vertices in the path to check the absence of loops. In such a case, we would need to consider an exponential number of paths.

Nevertheless, we will show that in the case where L belongs to $\mathcal{C}_{tract}$, we can identify a finite number of vertices that suffice to check if the path is (or can be transformed into) a simple path labeled with L. These "critical" vertices shall be stored in a path summary, as presented in the following. Unlike paths, summaries can be enumerated in logarithmic space, and we shall explain how one can use the summaries to check whether there exists a simple path between the input vertices. Roughly speaking, the idea of a summary is to keep only a bounded number of vertices of p, that depends only on L. Using Lemma 9, we actually show that it is enough to record the first vertex and the N last vertices having their state in C, for each component C of A_L. Additionally, if the number of such vertices is greater than $N+1$, we replace the path between the first vertex and the N last ones by a cut symbol cut_C. This symbol intuitively represents a Σ_C^*-labeled path that has been cut from the path. More formally, a summary is defined as follows.

DEFINITION 2 (LONG RUN COMPONENTS OF A PATH p). *Let $p = (v_1, a_1, \ldots, a_m, v_{m+1})$ be a path and let ρ be the run of L over p. A long run component of p is a component C of A_L such that there are at least $N + 2 = 2M^2 + 2$ vertices v in p such that $\rho(v) \in C$. We denote by $C_1, \ldots, C_l$ the long run components of p (the sequence is sorted by order of appearance in p). For each $i \in [l]$, Σ_{C_i} is the internal alphabet of C_i, $left_i$ is the first vertex v_j of p such that $\rho(v_j) \in C_i$ and $right_i$ is the last vertex v_j of p such that $\rho(v_j), \ldots, \rho(v_{j+N}) \in C_i$.*

DEFINITION 3 (CUT SYMBOLS AND SUMMARY). *We introduce a new "cut" symbol cut_C for each component C of A_L. The set of all cut symbols is denoted by Cuts. Let $p = (v_1, a_1, \ldots, a_m, v_{m+1})$ with run ρ. Let $(C_i, \Sigma_{C_i}, left_i, right_i)_{i \in [l]}$ be as defined in Definition 2. The summary S of the path p (w.r.t. A_L) is the sequence obtained from p by replacing, for each $i \in [l]$ the subpath $p[left_i, right_i]$ by the sequence $(left_i, cut_{C_i}, right_i)$ where cut_{C_i} is the cut symbol of component C_i.*

A summary contains at most $NM = 2M^3$ elements (vertices, labels and cut symbols), which is constant if L is fixed. Actually, we chose this large constant to simplify the presentation of the proofs and because we focus on data-complexity, considering L to be fixed. But the bound can easily be improved without even changing the structure of the proofs: in Lemma 8, $|C|^2$ is clearly an upper bound (not even tight) on the minimal length of synchronizing words. This allows to replace $N = 2M^2$ with $2M|C|$ in Lemma 9. This in turn allows to lower the size of a summary to $2M^2$.

As a consequence of this constant bound on the number of elements in a summary (for fixed L), each summary can be represented with a logarithmic number of bits.

We also observe that in a summary, all cut symbols are clearly distinct by definition of strongly connected components. A summary depends on a path p. However, we would like a notion of summary S that is independent of any path p. We thus define a candidate summary as an alternative sequence of vertices and symbols or cut symbols of the form above. Consequently, a summary is always a candidate summary but the converse is not true.

DEFINITION 4 (CANDIDATE SUMMARY). *We define as a candidate summary S any sequence of vertices and labels of the form above; $S = (v_1, \alpha_1, \ldots, \alpha_m, v_{m+1})$ where $\alpha_i \in \Sigma \cup Cuts$ for every $i \in [m]$, all cut symbols are distinct, and $m \leq NM$. Similarly to Definition 2, we denote by $cut_{C_1}, \ldots, cut_{C_l}$ the sequence of cut symbols appearing in S. Furthermore we define, for each $i \in [l]$, $left_i$ (resp. $right_i$) as the vertex at left (resp. right) of cut_{C_i} in p.*

A path p obtained by replacing each subsequence $(left_i, cut_{C_i}, right_i)$ with a simple $\Sigma_{C_i}^*$-labeled path from $left_i$ to $right_i$ is called a *completion* of the candidate summary S.

LEMMA 10. *Let S be the summary of an L-labeled path p and let p' be a completion of S. Then, p' is an L-labeled path with summary S.*

EXAMPLE 2. *Figure 2 represents the minimal DFA for $L = a(c^{\geq 2} + \epsilon)(a+b)^*(ac)?a^*$ (we did not represent the sink state). This automaton can loop in three strongly connected components: $C_1 = \{q_4\}$, $C_2 = \{q_5, q_6\}$, and $C_3 = \{q_7\}$. The accepting states are q_2, q_4, q_5, q_6, and q_7. We shall pretend that $N = 3$ for our example as this value is sufficient for our algorithm, although the correct value for N should be $2M^2$ according to our rough bounds. Let us consider the path p_1 illustrated in Figure 3 with thick edges. The table below details this path and the corresponding run.*

v_1	v_2	v_3	v_4	v_5	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
q_1	q_2	q_3	q_4	q_4	q_4	q_4	q_4	q_5	q_6	q_5	q_5	q_5	q_5
			run in C_1					*run in* C_2					

We observe that p_1 is a simple L-labeled path. The summary S of p_1 is obtained by removing the second (resp. second and third) vertex with state in C_1 (resp. C_2): only the highlighted vertices are preserved. The components of eliminated vertices are represented by cut symbols cut_{C_1} and cut_{C_2}.

$$S = (v_1, a, v_2, c, v_3, c, v_4, cut_{C_1}, v_7, c, v_8, c, v_9, a, v_{10}, cut_{C_2}, v_{13}, a, v_{14}, a, v_{15}).$$

Lemma 10 provides us with an NL algorithm to obtain an L-labeled path from a summary S. However, such a path is not necessarily simple, even if S is the summary

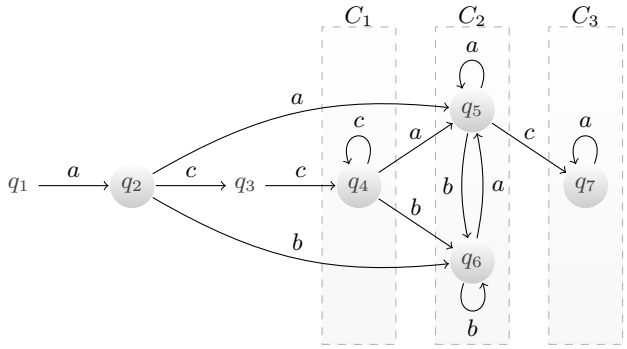


Figure 2: Minimal DFA for $a(c^{\geq 2}+\epsilon)(a+b)^*(ac)?a^*$

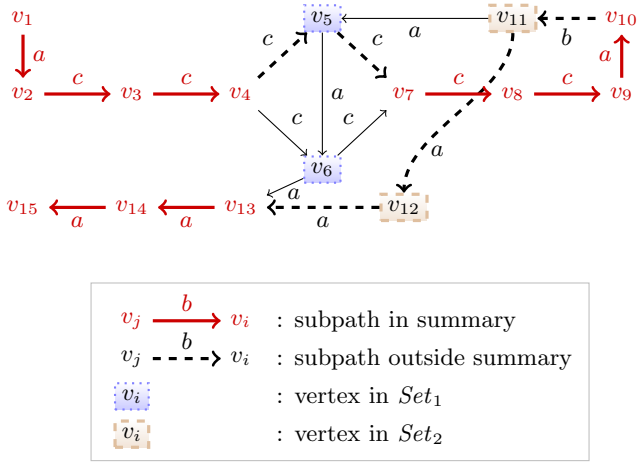


Figure 3: Admissible simple path, and its summary.

of a simple path. The reason is that the paths $(p_i)_{i\in[l]}$ we have built between each $left_i$ and $right_i$ are not necessarily disjoint. To overcome this problem, we will define *local domains* $Set_1, \ldots, Set_l$ which are disjoint sets of vertices from G. For each $i \in [l]$, we require the path p_i between $left_i$ and $right_i$ to be Set_i-restricted. Consequently, these paths will be disjoint. On the other hand, we do not want that the local domains are too restrictive because we must preserve the existence of at least one solution that satisfies these conditions for every positive instance (G, x, y) of RSPQ(L). In order to avoid oversized local domains, we limit Set_i to the set of vertices that might occur on a *shortest* $\Sigma_{C_i}^*$-labeled path from $left_i$ to $right_i$ that avoids all Set_j $(j < i)$. The following definition specifies more formally the local domains.

Definition 5 (local domains). *Let S be a candidate summary. Let $(C_i, left_i, right_i)_{i\in[l]}$ be as stated in Definition 4 and $V(S)$ be the set of vertices appearing in S. We define the local domains Set_i recursively for each i from 1 to l. The set Set_i will be defined as a particular subset of $V_i = V \setminus (V(S) \cup \bigcup_{j<i} Set_j)$. If there is no V_i-restricted $\Sigma_{C_i}^*$-labeled simple path p from $left_i$ to $right_i$, then $Set_i = \emptyset$. Otherwise, we denote by k_i the length of the shortest such path and define Set_i as the set of vertices y in V_i that can be reached from $left_i$ by a V_i-restricted $\Sigma_{C_i}^*$-labeled path p of length at most $k_i - 1$.*

The following lemma is a direct consequence of Definition 5.

Lemma 11. *Let S be a candidate summary. Then all sets $V(S), (Set_i)_{i\in[l]}$ from Definition 5 are disjoint.*

Definition 6 (admissible path). *Let p be a path with label in L, run ρ and summary S. We qualify p as* admissible *if the following two conditions are satisfied: (a) all vertices appearing in S are distinct and (b) for every $i \in [l]$, the path $p[left_i, right_i]$ is simple and Set_i-restricted.*

By the definition and Lemma 11, an admissible path is necessarily simple.

Lemma 12. *Let p be an admissible L-labeled path from x to y. Then p is simple.*

Example 3. *The path p_1 defined in Example 2 is admissible, since $Set_1 = \{v_5, v_6\}$ and $Set_2 = \{v_{11}, v_{12}\}$, as illustrated in Figure 3. The definition of admissible paths guarantees the paths replacing cut_{C_1} and cut_{C_2} are disjoint. Indeed we can check that $\{v_5\} \cap \{v_{11}, v_{12}\} = \emptyset$.*

Being an admissible path is clearly more restrictive than being a simple path. However, it turns out that shortest simple paths are admissible, as shown below. That means that the existence of a simple path is equivalent to the existence of an admissible path.

Lemma 13. *Let (G, x, y) be a RSPQ(L) instance. Every shortest simple L-labeled path from x to y is admissible.*

Proof. Let $p = (v_1, a_1, \ldots, a_m, v_{m+1})$ be a shortest simple L-labeled path from x to y. Assume that p is not admissible. That means there is some i and vertex v between $left_i$ and $right_i$ such that $v \notin Set_i$. We choose i minimal such that (1) there is $j > i$ and a vertex v between $left_j$ and $right_j$ such that $v \in Set_i$ or (2) there is a vertex v between $left_i$ and $right_i$ such that $v \notin Set_i$. For each of the two cases, we will construct a path p' shorter than p from x to y. We then prove that in both cases p' is an L-labeled simple path, which contradicts our assumption that p is the shortest such path.

case (1): Let us say that a vertex v satisfies property $\mathcal{P}_1$ if there exists $j > i$ such that $v \in Set_i$ and v appears in p between $left_j$ and $right_j$. Let v be a vertex satisfying property $\mathcal{P}_1$. Then, by definition of Set_i, there is a Set_i-restricted $\Sigma_{C_i}^*$-labeled simple path sp from $left_i$ to v that is shorter than the subpath $p[left_i, right_i]$ and, consequently, shorter than $p[left_i, v]$. Let p' be the path obtained from p by replacing $p[left_i, v]$ with sp. This path p' is shorter than p. For the remainder of the proof we assume that v is the last vertex in p satisfying property $\mathcal{P}_1$ and define p' accordingly.

case (2): we assume additionally that no vertex satisfies property $\mathcal{P}_1$. That means there is a vertex v' that does not belong to Set_i but belongs to $V \setminus (V(S) \cup \bigcup_{j<i} Set_j)$. Consequently, there is a Set_i-restricted L-labeled simple path sp between $left_i$ and $right_i$ that is no longer than $p[left_i, v']$ and, consequently, shorter than $p[left_i, right_i]$. We choose p' as the path obtained from p by replacing $p[left_i, right_i]$ with sp. Furthermore, for homogeneity of the proof, we define $v = right_i$.

The remainder of the proof is common to the two cases. We need to prove that p' is a simple L-labeled path. We first prove that p' is an L-labeled path. Let ρ' be the run of L over p'. Let w be the word formed by the labels of the

subpath $p[v, y]$. We know that $w \in \mathscr{L}_{\rho(v)}$ since p is an L-labeled path. We will show using Lemma 9 that $w \in \mathscr{L}_{\rho'(v)}$. By definition, $\rho'(left_i)$ belongs to the component C_i and the path $s = p'[left_i, v]$ is $\Sigma^*_{C_i}$-labeled, hence $\rho'(v) \in C_i$ by Lemma 7. Furthermore $\rho(v) \in C_j$ and C_j is reachable from C_i. In addition, by definition of a summary, there are at least $N+1$ vertices v'' of p after v such that $\rho'(v'') = \rho(v'') \in C_j$, and therefore the N labels following vertex v in p belong to Σ_{C_j} by Lemma 7 again. We have proved that $q_1 = \rho'(v)$ and $q_2 = \rho(v)$ meet all requirements for Lemma 9, which implies $w \in \mathscr{L}_{\rho'(v)}$. Consequently, p' is an L-labeled path.

We now prove that p' is simple. Since p is simple, it suffices to prove that the vertices in sp (between $left_i$ and v) are disjoint with other vertices in p'. Notice that all intermediate vertices of sp belong to Set_i. By minimality of i, for all $i' < i$, the vertices between $left_{i'}$ and $right_{i'}$ belong to $Set_{i'}$ and, since $Set_{i'}$ and Set_i are disjoint (Lemma 11), do not belong to Set_i. Consequently, there is no vertex v' before $left_i$ such that v' belong to Set_i. By construction, in the two cases (1) and (2), there is no vertex v' after v such that v' belongs to Set_i. That concludes the proof. □

We next show how an admissible summary can be completed in logarithmic space into a simple path. We first prove that local domains Set_i can be computed in logarithmic space.

LEMMA 14. *Let L be a fixed language in $\mathcal{C}_{tract}$. The following problem P_{Set} is in* NL. *Given an instance (G, x, y) of RSPQ(L), a candidate summary S, a vertex z and an integer i, decide whether $z \in Set_i$.*

PROOF. The proof is based on the following result due to Immerman [21]: $\mathrm{NL}^{\mathrm{NL}} = \mathrm{NL}$. In other words, if a decision problem P can be solved by an NL-algorithm using an oracle in NL, then this problem P belongs to NL [2]. Consider, for each $k \geq 0$, P^k_{Set} be the decision problem P_{Set} with the restriction $i \leq k$ i.e. (G, x, y, S, z, i) is a positive instance of P^k_{Set} iff (G, x, y, S, z, i) is a positive instance of P^k_{Set} and $i \leq k$. Notice that i belongs to the input of P^k_{Set} while this is not the case for k. Clearly, the number l of cuts in a summary S as in Definition 3, is bounded by the number K of strongly connected components of L. Consequently, $P_{Set} = P^K_{Set}$. Let us prove, that $P^k_{Set} \in \mathrm{NL}$ for each $k \geq 0$. If $k = 0$, P^0_{Set} always returns False because Set_i is not defined for $i = 0$. So P^0_{Set} is trivially in NL. Assume, by induction, that $P^k_{Set} \in \mathrm{NL}$. It suffices to show that there is an NL-algorithm for P^{k+1}_{Set} using P^k_{Set} as oracle. Since $\mathrm{NL}^{\mathrm{NL}} = \mathrm{NL}$, that implies that $P^{k+1}_{Set} \in \mathrm{NL}$. Let (G, x, y, S, z, i) be an instance of P^{k+1}_{Set}. If $i \leq k$, we return the same answer as the oracle P^k_{Set}. If $i = k + 1$, using the definition and notations of Set_{k+1}, the problem essentially boils down to computing the distances between the vertices $left_{k+1}$ and z on one side, $left_{k+1}$ and $right_{k+1}$ on the other side in the graph $G' = (V', E')$ where $V' = V_{k+1} \cup \{left_{k+1}, right_{k+1}\}$ and E' is the set of $\Sigma_{C_{k+1}}$-labeled edges of G. It is easily seen that this can be done in non deterministic log-space using the oracle P^k_{Set}. □

LEMMA 15. *Let L be a fixed language in $\mathcal{C}_{tract}$. There exists a non deterministic log-space algorithm that given an instance (G, x, y) of RSPQ(L) and a candidate summary S tests whether there is an admissible L-labeled path p from x to y with summary S.*

PROOF. We propose the following algorithm. It returns "Yes" if all the following tests succeed and "No" otherwise.

1. Check that all vertices of S are distinct;
2. Compute a path p from S by replacing, for each $i \in [l]$, the sequence $(left_i, cut_{C_i}, right_i)$ by a simple Set_i-restricted $\Sigma^*_{C_i}$-labeled path from $left_i$ to $right_i$. The test fails if it is not possible;
3. Check that p is an L-labeled path;
4. Check that S is the summary of p.

Let us prove that the algorithm is correct. First, we assume that there is an admissible (simple) L-labeled path with summary S from x to y. Then, by definition of an admissible path, test 1 and test 2 both succeed. Since p is a completion of S, Lemma 10 implies that p is an L-labeled path with summary S. Thus, tests 3 and 4 succeed too and the algorithm returns "Yes". Reciprocally, assume now that the algorithm returns "Yes". Tests 3 and 4 guarantee that p is an L-labeled path with summary S. Tests 1 and 2 guarantee that p is admissible. Consequently, p is an admissible L-labeled from x to y.

We still have to check the complexity. Notice that the sets Set_i are not stored in memory: we only need to check on-the-fly if a given vertex belongs those sets, which only requires logarithmic space according to Lemma 14. The same remark applies to the path p that is generated on the fly for the tests of tests 3 and 4. Taking these remarks into consideration, the algorithm can easily be implemented in NL. □

We eventually show the main Lemma of this section, proving that RSPQ(L) is tractable for every language in $\mathcal{C}_{tract}$.

LEMMA 16. *Let $L \in \mathcal{C}_{tract}$. Then, RSPQ($L$) $\in$ NL.*

PROOF. We simply enumerate all possible candidate summaries S w.r.t. (L, G, x, y), and apply on each summary the algorithm of Lemma 15. We return "Yes" if this algorithm returns "Yes" for at least one candidate summary S. Otherwise, we return "No". Therefore, our algorithm returns "Yes" if and only if there exists an admissible path from x to y, and consequently, if and only if there is a simple path from x to y (Lemmas 12 and 13). Since L is fixed, there is a polynomial number of candidate summaries, each of logarithmic size. Consequently, they can be enumerated within logarithmic space. □

Notice that we can easily adapt our algorithm so that it outputs a shortest path for positive instances. This computation of a shortest simple path generalizes to db-graphs weighted by a function $E \to \mathbb{R}^+$. The main theorem summarizes our results, combining Lemma 2 with Lemma 16.

THEOREM 1. *Let L be a regular language. Then, RSPQ(L) is in* NL *if $L \in trC$ and is* NP*-complete otherwise.*

3.3 Towards a complete classification

Actually, the classification can be made more precise. We have partitioned the RSPQ(L) problems into NL and NP-complete problems. To refine the partition, we now can envisage a classification within the class of NL problems.

[2]This result is true with the Ruzzo-Simon-Tompa model [37] but $\mathrm{NL}^{\mathrm{NL}} = \mathrm{NP}$ with others models of oracles.

LEMMA 17. *For every regular language L, RSPQ(L) $\in$ AC^0 if L is finite, otherwise RSPQ(L) is* NL*-hard.*

The proof is based on a reduction from the following NL-complete problem [34].

Reachability
Input: A directed graph G and two vertices x, y in G
Question: Is there a path from x to y?

PROOF. (Membership) Immerman [22] proves that AC^0 exactly corresponds to structures definable in FO (first order logic). Thus, we will prove that RSPQ(L) is definable in FO if L is finite. Given an alphabet Σ, we consider the signature $\tau = (R_a)_{a\in\Sigma}$ of binary predicates. We can view a db-graph (V, Σ, E) as a τ-structure $\mathcal{M} = (V, (R_a)_{a\in\Sigma})$ of domain V and such that for every $v_1, v_2 \in V$ and $a \in \Sigma$, $(v_1, v_2) \in R_a$ iff $(v_1, a, v_2) \in E$. Let $w = a_1 \dots a_k$ be a word. Let us define the predicate $\text{path}_w(x, y)$ that checks the existence of a simple w-labeled path between x and y. We let the reader verify that predicate $\text{path}_w(x, y)$ is expressible in FO.

(Hardness) We exhibit a reduction from Reachability. Let L be an infinite regular language. By the Pumping Lemma, there exist non empty words u, v, w such that $uv^*w \subseteq L$. We build a db-graph G' from G by first relabeling every edge of G with v, and then adding two vertices x' and y' with edges (x', u, x) and (y, w, y'). There is a (not necessarily simple) path from x to y in G iff there is an L-labeled simple path from x' to y' in G'. Consequently, RSPQ(L) is NL-hard. □

Our results so far can be summarized in the following trichotomy which refines Theorem 1.

THEOREM 2. *Let L be a regular language. The complexity of RSPQ(L) can be determined as follows.*

1. *L is finite: RSPQ(L) $\in$ AC^0;*
2. *$L \in \mathcal{C}_{tract}$ and L is infinite: RSPQ(L) is* NL*-complete;*
3. *$L \notin \mathcal{C}_{tract}$: RSPQ($L$) is* NP*-complete.*

3.4 Recognition of tractable languages

The following theorem establishes the complexity of deciding if RSPQ(L) is tractable (i.e. deciding if RSPQ(L) can be computed in polynomial time). We consider different representations of L (DFAs, NFAs and regular expressions).

THEOREM 3. *Testing whether a regular language L belongs to $\mathcal{C}_{tract}$ is:*

1. NL*-complete if L is given by a DFA;*
2. PSPACE*-complete if L is given by an NFA (resp. a regular expression).*

The proofs of hardness for DFAs and NFAs rely on reductions from the following two problems.

Emptiness
Input: A DFA $A_L = (Q_L, \Sigma, i_L, F_L, \Delta_L)$ that recognizes a language L
Question: is $L = \emptyset$?

Universality
Input: An NFA (or a regular expression) that recognizes a language $L \subseteq \{0, 1\}^*$
Question: $L = \{0, 1\}^*$?

The NL-completeness of Emptiness can easily be deduced from the NL-completeness of Reachability [34]. Stockmeyer and Meyer [40] proved that Universality is PSPACE-complete.

3.5 Characterization by regular expressions

In this section, we propose two characterizations of $\mathcal{C}_{tract}$ languages. The first in terms of regular expressions and the second in terms of a pumping property. Unlike the other properties discussed before on the minimal DFA of L, the pumping property is expressed directly on the language L. The languages in $\mathcal{C}_{tract}$ are exactly those that can be expressed with an expression in the fragment Ψ_{tr} defined below. This fragment essentially enforces restrictions on the concatenation of subexpressions: except at the highest level, only expressions of the form $e + \epsilon$ can be concatenated.

Ψ_{tr}-terms are defined as follows:

$$\Psi_{tr}\text{-term} \quad ::= \quad w + \epsilon \mid A^{\geq k} + \epsilon$$

where w is a word, A is a subset of Σ and $A^{\geq k}$ is a shortcut for $A^k A^*$. A Ψ_{tr}-sequence is a concatenation of terms $w\varphi_1 \dots \varphi_l w'$ where w and w' are words and $\varphi_1 \dots \varphi_l$ are Ψ_{tr}-terms. Finally, the fragment Ψ_{tr} is the set of disjunctions of Ψ_{tr}-sequences.

EXAMPLE 4. *For instance, the expression $a^*(b^{\geq 2} + \epsilon)c^*$ investigated in Example 1 belongs to the fragment Ψ_{tr} (using notation $c^* = c^{\geq 0}$). Expression $a^*ba^* + (a + b)^*$, on the opposite, does not, but is clearly equivalent to $(a + b)^{\geq 0}$, which does. The following theorem, however, implies that a^*ba^* is not equivalent to any expression from Ψ_{tr}.*

THEOREM 4. *Let L be a regular language. The three following statements are equivalent:*

1. *A language L belongs to $\mathcal{C}_{tract}$*
2. *L is recognized by a regular expression in Ψ_{tr}.*
3. *There is an integer $i \geq 0$ such that for every word $w_l, w_m, w_r \in \Sigma^*$ and every non empty words $w_1, w_2 \in \Sigma^+$, $w_l w_1^i w_m w_2^i w_r \in L$ implies $w_l w_1^i w_2^i w_r \in L$.*

COROLLARY 1. *$\mathcal{C}_{tract}$ is closed by intersection, union and word reversing.*

We observe that adapting the notion of summaries allows for a proof of Lemma 16 that directly considers regular expressions in Ψ_{tr}. Since $\mathcal{C}_{tract}$ is closed by union, we can restrict ourselves to Ψ_{tr}-sequences $\varphi_1 \dots \varphi_l$ where φ_1, φ_l are words and $\varphi_2, \dots, \varphi_{l-1}$ are Ψ_{tr}-terms. Let p be an L-path. We decompose p into subpaths $p_1, \dots p_l$ such that p_i matches the expression φ_i for every $i \in [l]$. The summary of p is built as follows:

- if φ_i is a word or an expression of the form $w + \epsilon$, we keep all vertices of p_i in the summary;
- if φ_i is an expression of the form $A^{\geq k} + \epsilon$, we keep the k first and k last vertices and replace the rest of the path by the special symbol cut_A.

4. OTHER RESULTS

This section investigates three further issues. First, we consider RSPQs over vertex-labeled graphs. Then, we give minor results on the parametrized complexity of the RSPQ problem. Finally, we discusses the complexity of RSPQs over graphs of bounded directed treewidth. These are straightforward applications of standard techniques, yet the results may be of practical interest.

4.1 Other models of database graphs

The goal of this section is to adapt our classification to another model of graphs: vl-graphs i.e. vertex-labeled graphs. We denote by vlg the class of vl-graphs.

For simplicity, we will consider vl-graphs as special db-graphs. We can put the label of a vertex into edges. Consequently, we see a vl-graph as a db-graph that respects the following restriction: there exists no pair of edges $e = (x, a, y)$ and $e' = (x', a', y)$ such that $a \neq a'$.

Clearly, given a language L, RSPQ(L, vlg) is at most as difficult as RSPQ(L). However, for some languages, the problem is easier. For example, RSPQ$(L, \text{vlg}) \in$ PTIME while RSPQ(L, vlg) is NP-complete for $L = a^*bc^*$. The key is that a vertex cannot have two different labels, and, consequently, a path that matches a^* is always disjoint from a path that matches c^*. By contrast, for $L = a^*ba^*$ or $L = (aa)^*$, the problem remains NP-complete.

By generalizing this, we can obtain the following result:

THEOREM 5. *Let $\mathcal{C}^{vlg}_{tract}$ be the class of regular languages L that satisfy $(aw_2)^M \mathscr{L}_{q_2} \subseteq \mathscr{L}_{q_1}$ for every pair of states $q_1, q_2 \in Q_L$, label a and pair of words w_1, w_2 such that $aw_1 \in Loop(q_1)$, $aw_2 \in Loop(q_2) \neq \emptyset$ and $q_2 \in \Delta_L(q_1, \Sigma^*)$.*

Let L be a regular language. Then, RSPQ(L, vlg) is in NL *if $L \in \mathcal{C}^{vlg}_{tract}$ and is* NP*-complete otherwise.*

4.2 Parametrized complexity

The next section focuses on the parametrized complexity of the RSPQ problem.

para-RSPQ$(\mathcal{L})$
Input: a db-graph graph $G = (V, \Sigma, E)$,
a regular language $L \in \mathcal{L}$ given by an NFA $A_L = (Q_L, i_L, F_L, \Delta_L)$
two vertices x and y
Parameter: $k = |Q_L|$
Question: Is there a simple L-path from x to y in G?

Our initial goal was to determine the parametrized complexity para-RSPQ$(\mathcal{C}_{tract})$. Unfortunately, we could only partially reach this goal. We first address the parametrized complexity of RSPQs when the parameter is the size of the path.

k-RSPQ
Input: a db-graph graph $G = (V, \Sigma, E)$,
a regular language L given by an NFA $A_L = (Q_L, i_L, F_L, \Delta_L)$,
two vertices x and y
an integer $k \geq 0$
Parameter: k
Question: Is there a simple L-labeled path of size at most k from x to y in G?

THEOREM 6. *k-RSPQ is* FPT*. More precisely, the problem is solvable in time $O(2^{O(k)}|A_L| \cdot |G| \cdot \log |G|)$.*

The proof is based on the Color Coding method [2]. As a consequence of this theorem we get:

COROLLARY 2. *Let $\mathcal{L}$ denote the class of finite languages. Then para-RSPQ$(\mathcal{L}) \in$* FPT*.*

The finite language can be given by an acyclic NFA or a star-free regular expression.

4.3 Directed treewidth

Directed treewidth is a notion introduced in [24]. It generalizes many other measures such as treewidth, dag-width or Kelly-width [8, 20]. Directed treewidth measures in some sense how close a digraph is to a DAG. Johnson et al. [24] present a general method to design polynomial algorithms on graphs of bounded directed treewidth. Like most algorithms exploiting treewidth, this method leverages a dynamic programming approach on the decomposition tree. They apply this method to show that testing the existence of a Hamiltonian path is polynomial on such classes of graphs. Here, we extend this result to show that the regular simple path problem is also computable in polynomial time for the same classes.

It has been observed in the literature that RSPQ has polynomial combined complexity on two interesting classes of graphs: graphs of bounded treewidth [7], and DAGs [30]. The result for DAGs is immediate indeed, as every path in a DAG is simple. The next theorem generalizes both these two results.

THEOREM 7. *Let $k \geq 0$ and $\mathcal{G}$ be a class of db-graphs with directed treewidth at most k. Then, RSPQ$(Reg, \mathcal{G})$ is polynomial, where Reg denotes the regular languages.*

5. CONCLUSION

We now pinpoint some directions for future work.

- As an extension of our work, we can consider context-free languages. It seems to be difficult to obtain useful results, since we can easily prove that distinguishing polynomial and NP-hard instances is undecidable if P $\neq$ NP (cf the long version [5]).
- We have studied the regular simple path problem from the data complexity perspective. An interesting continuation of our work is to include the language in the input (combined complexity). The question is to decide given a class of language $\mathcal{L}$ whether RSPQ$(\mathcal{L})$ is in P or NP-complete.
- What becomes tractable under restrictions to the graph such as planar digraphs or undirected graphs? Notice that both disjoint paths and even path problems are polynomial in these cases [26, 31, 35, 38].
- From the parametrized complexity perspective, what is the complexity of para-RSPQ$(\mathcal{C}_{tract})$? We conjecture that it is in FPT.

We would like to thank Thomas Schwentick, Slawomir Staworko, Sophie Tison and the anonymous referees for their thoughtful comments that let us improve the paper.

6. REFERENCES

[1] S. Abiteboul and V. Vianu. Regular path queries with constraints. *J. Comput. Syst. Sci.*, 58(3):428–452, 1999.

[2] N. Alon, R. Yuster, and U. Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995.

[3] M. Arenas, S. Conca, and J. Pérez. Counting beyond a yottabyte, or how sparql 1.1 property paths will prevent adoption of the standard. In *WWW*, pages 629–638, 2012.

[4] E. M. Arkin, C. H. Papadimitriou, and M. Yannakakis. Modularity of Cycles and Paths in Graphs. *J. ACM*, 38(2):255–274, 1991.

[5] G. Bagan, A. Bonifati, and B. Groz. A trichotomy for regular simple path queries on graphs. *CoRR*, abs/1212.6857, 2012.

[6] P. Barceló, L. Libkin, and J. L. Reutter. Querying graph patterns. In *PODS*, pages 199–210. ACM, 2011.

[7] C. L. Barrett, R. Jacob, and M. V. Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

[8] D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdrzálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012.

[9] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Answering regular path queries using views. In *ICDE*, pages 389–398, 2000.

[10] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. *J. Comput. Syst. Sci.*, 64(3):443–465, 2002.

[11] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

[12] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. An automata-theoretic approach to regular xpath. In *DBPL*, pages 18–35, 2009.

[13] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

[14] I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *SIGMOD Conference*, pages 323–330, 1987.

[15] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *ICDE*, pages 39–50. IEEE Computer Society, 2011.

[16] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. 2006.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[18] C. Gutierrez, C. A. Hurtado, A. O. Mendelzon, and J. Pérez. Foundations of semantic web databases. *J. Comput. Syst. Sci.*, 77(3):520–541, 2011.

[19] R. H. Güting. GraphDB: A Data Model and Query Language for Graphs in Databases. In *Proc. 20th Int. Conf. on Very Large Data Bases*, pages 297–308, 1994.

[20] P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008.

[21] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

[22] N. Immerman. *Descriptive complexity*. Springer, 1999.

[23] R. Jin, H. Hong, H. Wang, N. Ruan, and Y. Xiang. Computing label-constraint reachability in graph databases. In *SIGMOD Conference*, pages 123–134, 2010.

[24] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

[25] A. Koschmieder and U. Leser. Regular path queries on large graphs. In *SSDBM*, pages 177–194, 2012.

[26] A. S. Lapaugh and C. H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984.

[27] U. Leser. A query language for biological networks. In *ECCB/JBI*, page 39, 2005.

[28] L. Libkin and D. Vrgoc. Regular Path Queries on Graphs with Data. In *ICDT 2012*, pages 74–85, 2012.

[29] K. Losemann and W. Martens. The complexity of evaluating path expressions in sparql. In *PODS*, pages 101–112. ACM, 2012.

[30] A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.

[31] Z. P. Nedev. Finding an Even Simple Path in a Directed Planar Graph. *SIAM J. Comput.*, 29:685–695, 1999.

[32] Z. P. Nedev and P. T. Wood. A polynomial-time algorithm for finding regular simple paths in outerplanar graphs. *J. Algorithms*, 35(2):235–259, 2000.

[33] F. Olken. Graph data management for molecular biology. *OMICS*, 7(1):75–78, 2003.

[34] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[35] N. Robertson and P. D. Seymour. Graph Minors .XIII. The Disjoint Paths Problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[36] R. Ronen and O. Shmueli. Soql: A language for querying and creating data in social networks. In *ICDE*, pages 1595–1602, 2009.

[37] W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *J. Comput. Syst. Sci.*, 28(2):216–230, 1984.

[38] A. Schrijver. Finding k Disjoint Paths in a Directed Planar Graph. *SIAM J. Comput.*, 23(4):780–788, 1994.

[39] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[40] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

[41] C. B. Ward and N. M. Wiegand. Complexity results on labeled shortest path problems from wireless routing metrics. *Computer Networks*, 54(2):208–217, 2010.