FORMAL LANGUAGE CONSTRAINED PATH PROBLEMS

CHRIS BARRETT†, RIKO JACOB‡, AND MADHAV MARATHE†

Abstract. Given an alphabet Σ, a (directed) graph G whose edges are weighted and Σ-labeled, and a formal language $L \subseteq \Sigma^*$, the **Formal Language Constrained Shortest/Simple Path** problem consists of finding a shortest (simple) path p in G complying with the additional constraint that $l(p) \in L$. Here $l(p)$ denotes the unique word given by concatenating the Σ-labels of the edges along the path p. The main contributions of this paper include the following:

1. We show that the formal language constrained shortest path problem is solvable efficiently in polynomial time when L is restricted to be a context free language. When L is specified as a regular language we provide algorithms with improved space and time bounds.
2. In contrast, we show that the problem of finding any simple paths between a source and a given destination is **NP**-hard, even when L is restricted to fixed simple regular languages and to very simple classes of graphs (e.g. complete grids).
3. For the class of treewidth bounded graphs, we show that (i) the problem of finding a regular language constrained simple path between source and destination is solvable in polynomial time and (ii) the extension to finding context free language constrained simple paths is **NP**-complete.

Our results extend the previous results in [MW95, SJB97, Ya90]. Several additional extensions and applications of our results in the context of transportation problems are presented. For instance, as a corollary of our results, we obtain a polynomial time algorithm for the BEST k-SIMILAR PATH problem studied in [SJB97]. The previous best algorithm was given by [SJB97] and takes exponential time in the worst case.

Key words. Algorithms, Formal Languages, Computational Complexity, Transportation Planning, World Wide Web, Shortest Paths, Query Processing, Multi-criteria Problems.

AMS subject classifications. 68Q25, 68Q45, 90B06, 68R10

1. Introduction.

In many path finding problems arising in diverse areas, certain patterns of edge/vertex labels in the labeled graph being traversed are allowed/preferred, while others are disallowed. Thus, the feasibility of a path is determined by (i) its length (or cost) under well known measures on graphs such as distance, and (ii) its associated label. The acceptable label patterns can be specified as a formal language. For example, in transportation systems with mode options for a traveler to go from source to destination the mode selection and destination patterns of an itinerary that a route will seek to optimize can be specified by a formal language. The problem of finding label constrained paths also arises in other application areas such as production distribution network, VLSI Design, Databases queries [MW95, AMM97], etc. Here, we study the problem of finding shortest/simple paths in a network subject to certain formal language constraints on the labels of the paths obtained. We illustrate the type of problems studied here by discussing prototypical application areas:

1.1. Intermodal Route Planning.

Our initial interest in the problem studied in this paper came from our work in the **TRANSIMS**[1] project at the Los Alam-

†Email: {`barrett,marathe`}`@lanl.gov`. Los Alamos National Laboratory, P.O. Box 1663, MS M997, Los Alamos, NM 87545. Research supported by the Department of Energy under Contract W-7405-ENG-36.

‡Email: `rjacob@brics.dk`. BRICS, Basic Research in Computer Science, Center of the Danish Research Foundation. Department of Computer Science, University of Aarhus, Ny Munkegade, bg 540, DK-8000 Århus C, Denmark. Work done while staying at Los Alamos National Laboratory.

[1]TRANSIMS is an acronym for the "TRansportation ANalysis and SIMulation System". See `http://transims.tsasa.lanl.gov` for details.

os National Laboratory. We refer the reader to [TR+95a, AB+97, TR+95b] for a detailed description of this project.

As a part of the intermodal route planning module of **TRANSIMS**, our goal is to find feasible (near optimal) paths for travelers in an intermodal network (a network with several mode choices, such as train, car, etc.) subject to certain mode choice constraints. The mode choices for each traveler are obtained by either processing the data from the microsimulation module or by certain statistical models built from real life survey data. We refer the reader to the book by Ben-Akiva and Lerman [BaL] for a detailed discussion and references on the theory of discrete choice analysis as applied to transportation science. The following example illustrates a prototypical problem arising in this context.

Example 1: We are given a directed labeled, weighted, graph G. The graph represents a transportation network with the labels on edges representing the various modal attributes (e.g. a label t might represent a rail line). Suppose, we wish to find a shortest route from s to d for a traveler. This is the ubiquitous shortest path problem. But now, we are also told that the traveler wants to go from s to d using the following modal choices: either he walks to the train station, then uses trains and then walks to his destination (office), or would like to go all the way from home to office in his car. Using t to represent trains, w to represent walking and c to represent car, the travelers mode choice can be specified as $w^+t^+w^+ \cup c^*$, where $\cup, +$ and $*$ denote the usual operators used to describe regular languages.

1.2. Searching the Web. Browsing the Web to find documents of interest as well as searching a database using queries can be interpreted as graph traversals in a certain graph. From this viewpoint, one views a Web (database) as a directed (undirected), labeled graph—the nodes are URL sites (text) and edges are hyperlinks. A query for finding a particular URL site for instance, proceeds by browsing the network by following links and searching by sending information retrieval requests to "index servers". A serious problem that arises in the context of using pattern matching based search engines is that the queries cannot exploit the topology of the document network. For example as pointed out in [Ha88]:

> Content search ignores the structure of a hypermedia network. In contrast, structure search specifically examines the hypermedia structure for subnetworks that match a given pattern.

We refer the reader to the work of [MW95, AMM97, Ha88] for a more thorough discussion on this topic. A recent paper by Abiteboul and Vianu [AV99] also discusses how regular expression constrained path queries can be used to query the web. The following example is essentially from [AMM97].

Example 2: Let G be a graph describing a hypertext document. Suppose, we want to search for job opportunities for software engineers. We first query an index server to find pages that mention the keywords "employment job opportunities" and then, from each of these pages, we could follow the local paths of length zero, one or two to find pages that contain the keywords "software engineer". We can state the above problem as finding labeled paths in G with constraints on the admissible labelings. We refer to [AMM97] for a number of additional interesting queries that can be formulated in such a framework.

2. Problem Formulation. The problems discussed in the above examples can be formally described as follows: Let $G(V, E)$ be a (un)directed graph. Each edge $e \in E$ has two attributes — $l(e)$ and $w(e)$. $l(e)$ denotes the label of edge e. In this paper, the label is drawn from a (fixed) finite alphabet Σ. The attribute $w(e)$ denotes the weight of an edge. Here, we assume that the weights are non-negative integers. Most of our positive results can in fact be extended to handle negative edge weights also (if there are no negative cycles). A path p of length k from u to v in G is a sequence of edges $\langle e_1, e_2, \ldots e_k \rangle$, such that $e_1 = (u, v_1)$, $e_k = (v_{k-1}, v)$ and $e_i = (v_{i-1}, v_i)$ for $1 < i < k$. A path is *simple* if all the vertices in the path are distinct. Given a path $p = \langle e_1, e_2, \ldots e_k \rangle$, the weight of the path is given by $\sum_{1 \le i \le k} w(e_i)$ and the label of p is defined as $l(e_1) \cdot l(e_2) \cdots l(e_k)$. In other words the label of a path is obtained by concatenating the labels of the edges on the path in their natural order. Let $w(p)$ and $l(p)$ denote the weight and the label of p respectively.

DEFINITION 1. **Formal Language Constrained Shortest Path:** *Given a (un)directed labeled, weighted, graph G, a source s, a destination d and a formal language (regular, context free, context sensitive, etc.) L, find a shortest (not necessarily simple) path p in G such that $l(p) \in L$.*

DEFINITION 2. **Formal Language Constrained Simple Path:** *Given a (un)directed labeled, weighted, graph G, a source s, a destination d and a formal language (regular, context free, context sensitive, etc.) L, find a shortest simple path p in G such that $l(p) \in L$.*

For the rest of the paper we denote the formal language constrained shortest path problem restricted to regular, context free and context sensitive languages by REG-SHP, CFG-SHP and CSG-SHP respectively. Similarly, we denote the formal language constrained simple path problem restricted to regular, context free and context sensitive languages by REG-SIP, CFG-SIP and CSG-SIP respectively.

In general we consider the input for these problems to consist of a description of the graph (including labeling and weights) together with the description of the formal language as a grammar. By restricting the topology of the graph and/or the syntactic structure of the grammar we get modifications of the problems. If we claim a statement to be true 'for a *fixed* language' we refer to the variant of the problem, where the input consists of the graph only, whereas the language is considered to be part of the problem specification.

Note that in unlabeled networks with non-negative edge weights, a shortest path between s and d is necessarily simple. This need not be true when we wish to find a shortest path subject to an additional constraints on the set of allowable labels. As a simple example, consider the graph $G(V, E)$ that is a simple cycle on 4 nodes. Let all the edges have weight 1 and label a. Now consider two adjacent vertices x and y. The shortest path from x to y consists of a single edge between them; in contrast a shortest path with label $aaaaa$ consists of a cycle starting at x and the additional edge (x, y).

3. Summary of Results:. We investigate the problem of formal language constrained path problems. A number of variants of the problem are considered and both polynomial time algorithms as well as hardness results (**NP**-, **PSPACE**-hardness, undecidability) are proved. Two of the **NP**-hardness results are obtained by combining the simplicity of a path with the constraints imposed by a formal language. We believe that the techniques used to prove these results are of independent interest. The main results obtained in the paper are summarized in Figure 1 and include the following:

1. We show that CFG-SHP has a polynomial time algorithm. For REG-SHP with operators $(\cup, \cdot, *)$, we give polynomial time algorithms that are substantially more efficient in terms of time and space. The polynomial time solvability holds for the REG-SHP problem, when the underlying regular expressions are composed of

$(\cup, \cdot, *, 2)^\dagger$ operators. We also observe that the extension to regular expressions over operators $(\cup, \cdot, *, -)$ is **PSPACE**-hard.

2. In contrast to the results for shortest paths, we show that the problem finding any *simple* paths between a source and a given destination is **NP**-hard, even when restricted to a very simple, fixed locally testable regular language (see Section 6.2 for details), and very simple graphs (undirected grid graphs).

3. In contrast to the results in (1) and (2) above, we show that for the class of treewidth bounded graphs, (i) the REG-SIP is solvable in polynomial time, but (ii) CFG-SIP problem is **NP**-complete, even for a fixed deterministic linear context free language. The easiness proof can be extended to one-way-log-space recognizable languages. It uses a dynamic programming method; although the tables turn out to be quite intricate.

4. Finally, we investigate the complexity of the problems CSG-SHP and CSG-SIP. Using simple reductions and in contrast to the complexity results in (1) and (2), we show that (i) CSG-SIP is **PSPACE**-complete but (ii) CSG-SHP is *undecidable* even for a fixed language.

5. As an application of the theory developed here, we provide a polynomial time algorithm for a number of basic problems in transportation science. In Section 7 we consider two examples, namely the BEST k-SIMILAR PATH and the TRIP CHAINING problem.

The results mentioned in (1)–(4) provide a tight bound on the computational complexity (**P** versus. **NP**) of the problems considered, given the following assumptions: The inclusions of classes of formal languages "finite $\subset$ locally testable" and "regular $\subset$ deterministic linear context free" are tight, i.e. there is no natural class of languages "in between" these classes. Furthermore in this paper grid-graphs are considered to be the "easiest" class of graphs that do not have a bounded treewidth.

Preliminary versions of the algorithms outlined here have already been incorporated in the Route Planning Module of **TRANSIMS**. In [JMN98] we conduct an extensive empirical analysis of these and other basic route finding algorithms on realistic traffic network.

4. Related Work. We refer the reader to the monograph by Huckenbeck [Hu97] for a comprehensive survey on path problems. References [TR+95a, AB+97, TR+95b] provide a detailed account of the **TRANSIMS** project. Regular expression constrained simple path problems were considered by Mendelzon and Wood [MW95]. The authors investigate this problem in the context of finding efficient algorithms for processing database queries (see [CMW87, CMW88, MW95]). A recent paper by Abiteboul and Vianu describes further results on related problems [AV99]. Yannakakis [Ya90] in his keynote talk has independently outlined some of the polynomial time algorithms given in Section 5. Romeuf [Ro88] also independently considered some of the problems discussed in Section 5. However, the emphasis in [Ya90] was on database theory and Romeuf [Ro88] only considered regular languages. Online algorithms for regular path finding are given in [BKV91]. Our work on finding formal language constrained shortest paths is also related to the work of Ramalingam and Reps [RR96]. The authors were interested in finding a minimum cost derivation of a terminal string from one or more non-terminals of a given context free grammar. The problem was first considered by Knuth [Ku77] and is referred to as the *grammar problem*. [RR96] give an incremental algorithm for a version of the grammar problem

---

†operator $\square^2$ or simple 2 stands for the square operator. R^2 denotes $R \cdot R$.

	word recognition	shortest path general graph	simple path treewidth bound	simple path grid graph
fixed finite	**FP**	**FP**	**FP**	**FP**
free finite	**FP**	**FP**	**FP**	**NP**-c.
fixed LT	**FP**	**FP**	**FP**	**NP**-c.[5]
free RL	**FP**	**FP**[1]	**FP**	**NP**-c.
fix. 1-log-SPCE-TM	**FP**	undec[2]	**FP**	**NP**-c.
fix. lin. det. CFL	**FP**	**FP**	**NP**-c.[4]	**NP**-c.
free CFL	**FP**	**FP**[3]	**NP**-c.	**NP**-c.
fixed CSL	**PSPACE**-c.	undec.	**PSPACE**-c.	**PSPACE**-c.

[1]Section 5.1, Theorem 11; [2]Section 7.5; [3]Section 5.3; [4]Section 6.3, Theorem 26; [5]Section 6.2, Theorem 20.

FIG. 1. *Summary of results on formal language constrained simple/shortest paths in contrast to the word recognition problems. LT, RL, CFL, CSL denote locally testable, regular, context free and context sensitive languages respectively. For regular languages, the time bounds hold for regular expressions with the operators* $(\cup, \cdot, *, 2)$. **FP** *states that the problem can be computed in deterministic polynomial time, even if the language specification is part of the input. The superscripts in the table and the corresponding text tell where the result is proven.*

and as corollaries obtain incremental algorithms for single source shortest path problems with positive edge weights. We close this section with the following additional remarks:

1. To our knowledge this is the first attempt to use formal language theory in the context of modeling mode/route choices in transportation science.

2. The polynomial time algorithms and the hardness results presented here give a boundary on the classes of graphs and queries for which polynomial time query evaluation is possible. In [MW95] the authors state

> Additional classes of queries/dbgraphs for which polynomial time evaluation is possible should be identified ...

Our results significantly extend the known hardness as well as easiness results in [MW95] on finding regular expression constrained simple paths. For example, the only graph theoretic restriction considered in [MW95] was acyclicity. On the positive side, our polynomial time algorithms for regular expression constrained simple path problem when restricted to graphs of bounded treewidth are a step towards characterizing graph classes on which the problem is easy. Specifically, it shows that for graphs with fixed size recursive separators the problems is easy. Examples of graphs that can be cast in this framework include chordal graphs with fixed clique size, outer planar graphs, series parallel graphs, etc. (see [Bo92] for other examples).

3. The basic techniques extend quite easily (with appropriate time performance bounds) to solve other (regular expression constrained) variants of shortest path problems. Two notable examples that frequently arise in transportation science and can be solved are (i) multiple cost shortest paths [Ha92] and (ii) time dependent shortest paths [OR90]. These extensions are briefly outlined in Section 7.

The rest of the paper is organized as follows. Section 4.1 contains preliminary results and basic definitions. In Section 5, we present efficient algorithms for grammar constrained shortest path problems (namely, REG-ShP and CFG-ShP). Section 6 contains our hardness/easiness results for simple paths. Section 7 outlines several extensions and applications of our basic results.

4.1. Basic Definitions. We recall the basic concepts in formal language and graph theory. Additional basic definitions on topics related to this paper can be found in [HU79, GJ79, AHU, CLR]. For the rest of the paper, we use $|I|$ to denote the size of an object I represented using Binary notation.

DEFINITION 3. *Let Σ be a finite alphabet disjoint from $\{\varepsilon, \phi, (,), \cup, \cdot, *\}$. A regular expression R over Σ Is defined as follows:*

1. *The* empty string *"ε", the* empty set *"ϕ" and for each $a \in \Sigma$, "a" are atomic regular expressions.*
2. *If R_1 and R_2 are regular expressions, then $(R_1 \cup R_2)$, $(R_1 \cdot R_2)$ and $(R_1)^*$ are compound regular expressions.*

DEFINITION 4. *Given a regular expression R, the language (or the set) defined by R over Σ and denoted by $L(R)$ is defined as follows.*

1. $L(\varepsilon) = \{\varepsilon\}$; $L(\phi) = \phi$; $\forall a \in \Sigma$: $L(a) = \{a\}$
2. $L(R_1 \cup R_2) = L(R_1) \cup L(R_2) = \{\, w \mid w \in L(R_1)$ *or* $w \in L(R_2)\,\}$
3. $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2) = \{\, w_1 w_2 \mid w_1 \in L(R_1)$ *and* $w_2 \in L(R_2)\,\}$
4. $L(R^*) = \bigcup_{k=0}^{\infty} L(R)^k$ *where* $L(R)^0 = \{\varepsilon\}$ *and* $L(R)^i = L(R)^{i-1} \cdot L(R)$

DEFINITION 5. *A* nondeterministic finite automaton *(NFA) is a 5-tuple $M = (S, \Sigma, \delta, s_0, F)$, where*

1. *S is a finite nonempty set of states;*
2. *Σ is the input alphabet (a finite nonempty set of letters);*
3. *δ is the* state transition function *from $S \times (\Sigma \cup \{\varepsilon\})$ to the power set of S;*
4. *$s_0 \in S$ is the initial state;*
5. *$F \subseteq S$ is the set of accepting states.*

If there is an $s \in S$ such that $\delta(s, \varepsilon)$ is a nonempty subset of S, then the automaton M is said to have ε-transitions. If M does not have any ε-transitions and for all $s \in S$ and $a \in \Sigma$ the set $\delta(s, a)$, has at most one element, then δ can be regarded as a (partial) function from $S \times \Sigma$ to S and M is said to be a *deterministic finite automaton* (DFA).

The extended transition function δ^* from $S \times \Sigma^*$ is defined in a standard manner. The size of M denoted by $|M|$ is defined to be equal to $|S||\Sigma|$.

DEFINITION 6. *Let $M = (S, \Sigma, \delta, s_0, F)$ be a NFA. The* language accepted *by M denoted $L(M)$, is the set*

$$L(M) = \{\, w \in \Sigma^* \mid \delta^*(s_0, w) \cap F \neq \phi \,\}$$

A string w is said to be accepted by the automaton M if and only if $w \in L(M)$.

DEFINITION 7. *A* context free grammar *(CFG) G is a quadruple (V, Σ, P, S), where V and Σ are disjoint nonempty sets of* nonterminals *and* terminals*, respectively, $P \subset V \times (V \cup \Sigma)^*$ is a finite set of* productions *and S is the* start symbol. *A CFG G is said to be* linear *if at most one nonterminal appears on the right-hand side of any of its productions.*

DEFINITION 8. *[Bo88, AL+91] Let $G = (V, E)$ be a graph. A* **tree-decomposition** *of G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a family of subsets of V and $T = (I, F)$ is a tree with the following properties:*

1. $\bigcup_{i \in I} X_i = V$.
2. *For every edge $e = (v, w) \in E$, there is a subset X_i, $i \in I$, with $v \in X_i$ and $w \in X_i$.*
3. *For all $i, j, k \in I$, if j lies on the path from i to k in T, then $X_i \bigcap X_k \subseteq X_j$.*

The **treewidth** *of a tree-decomposition* $(\{X_i \mid i \in I\}, T)$ *is* $\max_{i \in I} |X_i| - 1$. *The treewidth of* G *is the minimum treewidth of a tree decomposition.*

5. Shortest Paths. In this section, we present polynomial time algorithms for the problems REG-ShP and CFG-ShP.

5.1. Algorithm for REG-ShP and Extensions. In this subsection, we will describe our algorithms for regular expression constrained shortest path problems. We note that regular expressions over $(\cup, \cdot, *)$ can be transformed into equivalent NFAs in $O(n)$ time [AHU], where n represents the size of the regular expression. Thus for the rest of this subsection we assume that the regular expressions are specified in terms of an equivalent NFA.

The basic idea behind finding shortest paths satisfying regular expressions is to construct an auxiliary graph (the product graph) combining the NFA denoting the regular expression and the underlying graph. We formalize this notation in the following.

DEFINITION 9. *Given a labeled directed graph* G*, a source* s *and a destination* d*, define the NFA* $M(G) = (S, \Sigma, \delta, s_0, F)$ *as follows:*

1. $S = V$; $s_0 = s$; $F = \{d\}$;
2. Σ *is the set of all labels that are used to label the edges in* G *and*
3. $j \in \delta(i, a)$ *iff there is an edge* (i, j) *with label* a.

Note that this definition can as well be used to interpret an NFA as a labeled graph.

DEFINITION 10. *Let* $M_1 = (S_1, \Sigma, \delta_1, p_0, F_1)$*, and* $M_2 = (S_2, \Sigma, \delta_2, q_0, F_2)$*, be two NFAs. The product NFA is defined as* $M_1 \times M_2 = (S_1 \times S_2, \Sigma, \delta, (p_0, q_0), F_1 \times F_2)$*, where* $\forall a \in \Sigma$, $(p_2, q_2) \in \delta((p_1, q_1), a)$ *if and only if* $p_2 \in \delta_1(p_1, a)$ *and* $q_2 \in \delta_2(q_1, a)$.

It is clear that $L(M_1 \times M_2) = L(M_1) \cap L(M_2)$. ALGORITHM RE-CONSTRAINED-SHORT-PATHS outlines the basic steps for solving the problem and uses the cross product construction mentioned above.

ALGORITHM RE-CONSTRAINED-SHORT-PATHS:

- *Input:* A regular expression R, a directed labeled weighted graph G, a source s and a destination d.
- 1. Construct an NFA $M(R) = (S, \Sigma, \delta, s_0, F)$ from R.
  2. Construct the NFA $M(G)$ of G.
  3. Construct $M(G) \times M(R)$. The length of the edges in the product graph is chosen to be equal to the corresponding edges in G.
  4. Starting from state (s_0, s), find a shortest path to the vertices (f, d), where $f \in F$. Denote these paths by p_i, $1 \le i \le w$. Also denote the cost of p_i by $w(p_i)$
  5. $C^* := \min_{p_i} w(p_i)$; $\quad p^*$: $w(p^*) = C^*$.
     (If p^* is not uniquely determined, we choose an arbitrary one.)
- *Output:* The path p^* in G from s to d of minimum length subject to the constraint that $l(p) \in L(R)$.

THEOREM 11. *The Algorithm* RE-CONSTRAINED-SHORT-PATHS *computes the exact solution for the problem* REG-ShP *with non-negative edge weights in time* $O(T(|R||G|))$. *Here* $T(n)$ *denotes the running time of a shortest path algorithm on a graph with* n *nodes.*

Proof. For the correctness of the algorithm observe the following: Consider a shortest path q^* in G of cost $w(q^*)$, that satisfies the regular expression R. This path and the accepting sequence of states in the NFA imply a path q' of the same cost between (s_0, s) to (f, d) for some $f \in F$. So we know that the sought path is considered. Conversely, for each path τ in $M(G) \times M(R)$ of cost $w(\tau)$, that begins on a starting state and ends on a final state, the projection of τ to the nodes of G yields a path of same cost in G from s to d that satisfies the regular expression R.

To calculate the running time of the algorithm, observe that the size of $M(R) \times M(G)$ is $O(|R||G|)$. As the overall running time is dominated by Step 4, we obtain $O(T(|R||G|))$ as a bound. □

5.2. Extensions: Other Regular Expressions. We consider two possible extensions for the problem REG-SHP, namely allowing the additional operators for taking the complement $(-)$ and for squaring an expression (2).

THEOREM 12. *Shortest Path in the complement of an NFA or a Regular Expression over* $(\cup, \cdot, *)$ *is* **PSPACE**-*hard.*

Proof. Let R be such a regular expression. The question of deciding if the complement of $L(R)$ is empty, (REGULAR EXPRESSION NON-UNIVERSALITY) is known to be **PSPACE**-complete. ([GJ79], problems AL1 and AL9). Given an instance of such a problem we create a graph G with one node v. There is a loop from v to v for each symbol in the alphabet. The existence of a path from v to v with the label in the complement of the language $(\Sigma^* - L(R))$ is equivalent to the existence of such a word. □

This immediately implies

COROLLARY 13. REG-SHP *for regular expressions over* $(\cup, \cdot, *, -)$ *is* **PSPACE**-*hard.*

It is easy to see that regular expressions consisting of operators from $(\cup, \cdot, *, 2)$ can be represented by context free grammars with rules of the form $A \to BB$. This observation together with the results of the next section (Section 5.3) yields the following:

COROLLARY 14. REG-SHP *for regular expressions over* $(\cup, \cdot, *, 2)$ *can be solved in polynomial time.*

5.3. Algorithm for CFG-SHP. We now extend our results in Section 5.1 to obtain polynomial time algorithms for context free language constrained shortest path problems. Beside the applicability of the algorithm, this result stands in contrast to the hardness of simple path problems even for a single fixed regular expression.

The algorithm for solving context free grammar constrained shortest paths is based on dynamic programming. Hence we will first investigate the structure of an optimal shortest path from s to d in the graph G that is labeled according to the context free Grammar R. Assume that R is in Chomsky normal form, i.e. all rules of the form $C \to AB$ or $C \to a$ (see [HU79] for details). Consider any such shortest path p with $l(p) = a_1 a_2 \ldots a_m$. One important property of any CFG is that nonterminals are expanded independently. In the case of a Chomsky Normal form, the derivation forms a binary tree, which means that the label of p can be decomposed into two parts l_1 and l_2 such that $l(p) = l_1 l_2$, $S \to AB$, $A \xrightarrow{*} l_1$ and $B \xrightarrow{*} l_2$.

With this structure in mind let us define the quantity $D(i, j, A)$ as the shortest path distance from i to j subject to the constraint that the label on this path can be derived starting from the nonterminal A.

These values are well defined and fulfill the following recurrence:

$$D(i,j,A) = \min_{(A\to BC)\in R} \min_{k\in V} \Big(D(i,k,B) + D(k,j,C) \Big) \tag{1}$$

$$D(i,j,a) = \begin{cases} w(i,j) & \text{if } l((i,j)) = a; \\ \infty & \text{otherwise.} \end{cases} \tag{2}$$

OBSERVATION 15. *These equations uniquely determine the function D and immediately imply a polynomial time dynamic programming algorithm.*

Proof. Consider the case, that D and D' satisfy the above recurrence but are different. Then there must be a smallest witness of this fact $a = D(i,j,X)$ and $b = D'(i,j,X)$, and $a \neq b$. Let us assume $a > b$ and consider the smallest such b. As the definition (2) is unambiguous, we know that $X = A$ is a nonterminal. As b is minimal, it is finite and satisfies (1). This implies that there must exist the witnesses $e = D(i,k,B)$ and $f = D(k,j,C)$ that establish $b = e+f$ as their sum. As all lengths in the graph are positive, both e and f are smaller than b. By the choice of a and b we know that $e = D'(i,k,B)$ and $f = D'(k,j,C)$, implying with (1) the contradiction $a \leq e + f = b$.

This discussion immediately implies a dynamic programming approach to compute the table of D. Starting with the values for links in the network, we fill the table with increasing values. This can be done with a Bellman-Ford type algorithm. This algorithm will finally set at least one entry in the table per round. The execution time is bound by the square of the number of entries in the table times the amount of time needed to compute the two minima in 1. This is polynomial, namely $O(|V|^5|N|^2|R|^2)$ with V being the vertices of the graph, N the nonterminals and R the rules of the grammar in Chomsky normal form.

Another way of implementing this is to set the table by filling in the smallest values first. There a heap is used to hold the current estimates on entries, and the smallest one is finally put in the table generating or changing estimates. This implies for every entry one extract-min operation and up to $2|V||R|$ update operations. Using Fibonacci-Heaps (see [CLR] for an analysis), this sums up to $O(|V|^2|N| \cdot (\log(|V|^2|N|) + 2|V||R|))$, that is $O(|V|^3|N||R|)$. ☐

A naive adaptation of a Floyd-Warshall type algorithm fails, because we cannot split an optimal path in two optimal subsolutions at an arbitrary node of the path. The splitting in the above dynamic programming works only because it is done in accordance with the grammar.

6. Simple Paths. Next, we investigate the complexity of finding formal language constrained simple paths. For the ease of exposition, we present our results for directed, multilabeled graphs. The following lemma shows how to extend these results to undirected and unilabeled graphs.

LEMMA 16.

1. REG-SIP *on directed, multilabeled grids can be reduced to* REG-SIP *on directed grids.*
2. REG-SIP *on directed grids can be reduced to* REG-SIP *on undirected grids.*
3. *for all $k \geq 1$,* CFG-SIP *on directed graphs of treewidth k can be reduced to* CFG-SIP *on undirected graphs of the same treewidth.*

Proof. **(1)** Applying the changes illustrated in Figure 2 to all nodes of a multilabeled grid G, we obtain an unilabeled grid G'. G' is part of a grid roughly $|\Sigma|^2$ times larger than the original one. Furthermore, the alphabet needs to be extended by two symbols, one for inaccessible edges and the symbol v for the edges “inside” the extended nodes.

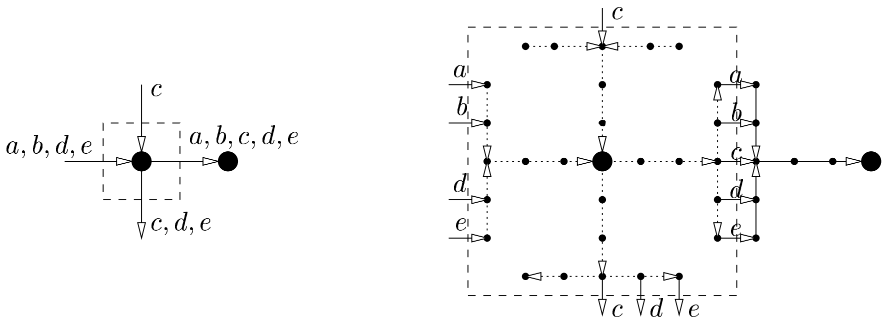

FIG. 2. *Example for replacing multi-labels. All the dotted edges “inside” are labeled with v. Only the connection to the right is completely depicted.*

The regular language L is replaced by the regular language L' defined as follows:

$$L' = \{\ w \in (\Sigma \cup \{v\})^* \mid w = v^* x_1 v^* x_2 v^* \cdots v^* x_n v^* \ \text{ and } \ x_1 x_2 \cdots x_n \in L\ \}$$

Now paths in the new instance (G', L') are in one to one correspondence to paths in the original instance (G, L).

(2), (3) It is well known (see for example [HU79]) that regular and context free languages are closed under substitution. Let $\Sigma = \{a, b, \ldots, z\}$ be an alphabet. Then $\Sigma' = \{a', b', \ldots, z'\}$ is a marked copy of this alphabet. Substitution with the homomorphism $a \mapsto aa'$ for all $a \in \Sigma$ yields the following: If $L \subseteq \Sigma^*$ is regular, the language

$$L' = \{\ w \in (\Sigma \cup \Sigma')^* | w = x_1 x_1' x_2 x_2' \cdots x_n x_n' \ \text{ and } \ x_1 x_2 \cdots x_n \in L\ \}$$

is also regular. Let G be the original graph. The directed edges in G labeled with a, get replaced by two consecutive edges labeled with a and a', introducing a new node.

In this situation paths in G' complying with L' are in a one-to-one correspondence to paths in G complying with L. It is straightforward to extend the weight function on the edges to preserve the weights of paths. Additionally relative path length (in number of edges used) are preserved.

The proof of (2) follows from the fact that the resulting G' can be embedded in a grid using a new symbol $v \notin (\Sigma \cup \Sigma')$ as label.

For (3) let T be a tree-decomposition of treewidth k. If $k = 1$ the graph itself is a tree and replacing edges by paths of length two does not change the treewidth. Otherwise let T be a tree-decomposition of width $k > 1$. For every new node we create a new set of the tree-decomposition consisting of the new node and the endpoints of the edge it splits. This set is included in the tree of the decomposition by attaching it to the set that covered the split edge. As $k > 1$ sets of cardinality three cannot change the treewidth, yielding a new tree-decomposition of width k. $\square$

6.1. Finite Languages. DEFINITION 17. *A language L is called* finite *if there are only finitely many words in L.*

Finite languages are considered one of the smallest subclasses of the regular languages.

THEOREM 18. *For any* fixed *finite language L the problem* REG-SIP *can be solved in polynomial time.*

Proof. Let k be the maximum length of a word in L. Considering all k-tuples of nodes, and checking if they form the sought path, yields a polynomial time algorithm of running time $O(n^k)$. □

THEOREM 19. *Let $\mathcal{C}$ be a graph class (such as planar, grid, etc) such that the* HAMILTONIAN PATH *problem is* **NP***-hard when restricted to $\mathcal{C}$. Then the problem* REG-SIP *is* **NP***-hard when restricted to $\mathcal{C}$ and (free) finite languages.*

Proof. Consider a fixed class of graphs $\mathcal{C}$ for which the HAMILTONIAN PATH problem is **NP**-hard. Then given an instance G of the HAMILTONIAN PATH problem in which $G \in \mathcal{C}$, with n nodes, we construct an instance G_1 of the regular expression constrained simple path problem by labeling all the edges in G by a. We now claim that there is a Hamiltonian path in G if and only if there is a simple path in G_1 that satisfies a^{n-1}, i.e. the constraining language is chosen to be the finite language $L = \{a^{n-1}\}$. □

6.2. Hardness of REG-SIP**.** Before formally stating the theorem and the proof we present the overall idea. We perform a reduction from 3-SAT. It will be easy to verify that the reduction can be carried out in polynomial time (logarithmic space).

Our encoding of a satisfiability question naturally decomposes into two parts. The first is to choose an assignment, the second is to check whether this assignment satisfies the given formula. Choosing the assignment will correspond to choosing a certain part of the path. The 3-CNF formula will be checked clause by clause. This requires access to the value of a variable more than once. Here this is achieved by forcing sufficiently many subpaths to be similar (in the sense that they stand for the same assignment). These "copies" of the assignment can then be used to check whether the underlying assignment satisfies the 3-CNF formula. This is now a local task, as there is one copy of the assignment for every clause.

In order to achieve sufficiently many similar copies of the assignment, the following *beads and holes argument* will be useful: Think of $n + 1$ holes forming a straight line and n beads between every two of them. Each bead is allowed to fall in one of the two holes adjacent to it. Moreover, we allow at most one bead in each hole. The state of the system after the beads fall down in the holes is a description of the contents of each hole. By reporting which hole is free, the state of the beads and holes system is described completely; since all beads left of the free hole fell in the left hole and the remaining beads fell into the right hole. We additionally know that there exists a set of at least $n/2$ consecutive beads that fell in the same direction.

The construction presented enforces an overall snake like path, that goes up and down several times. This is schematically depicted in Figure 3. Figure 4 provides additional details about the construction and will be described in the sequel. At this point it is sufficient to note that nodes on a vertical dotted line will be referred to as nodes on a column and vertices on each horizontal dotted line will be referred to as nodes on a level (or a row). For the case of n levels (variables) and m columns the following statements provide an overall outline of the proof:

1. Every column of the rotated grid (depicted in Figure 4) represents a member of an extended sequence of clauses. Any feasible path from S to D will use all of the column nodes. Such a path naturally decomposes into subpaths which span an entire column and are called legs.

2. The shape of a leg uniquely corresponds to an assignment; for this levels of the rotated grid are identified with variables.

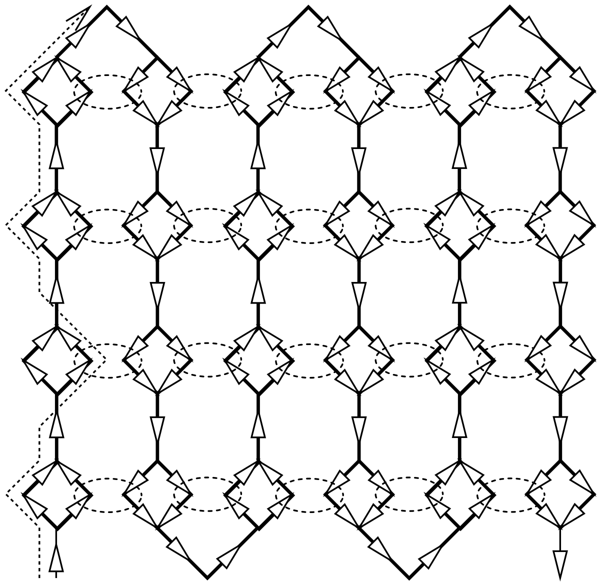

FIG. 3. *Overall shape of all feasible paths; the dotted circles stand for nodes to be identified. One possible leg in the first column is depicted as a dashed line.*

3. Identically shaped legs do not interfere on neighboring columns.
4. Let p be one fixed feasible path, i.e. a path that consists of m legs, as many as there are columns. Such a path visits every node but precisely n variable nodes, one per level. Therefore the shape of the legs of p cannot change too often (following the beads and holes argument), which in turn assures that there will be a lot of similar legs.
5. On each column the labeling together with the language allows us to check that the assignment represented by the leg satisfies a clause (of the extended set of clauses). For this we use three symbol of the alphabet to stand for the three literals of the clause.

We will prove the theorem for multi-labeled directed grid-graphs. Lemma 16 implies that the result holds for unilabeled undirected grid graphs.

THEOREM 20. *The* REG-SIP *Problem is* **NP***-hard for complete multi-labeled directed grids and a* fixed *regular expression.*

Proof. We present a reduction from the problem 3-SAT, which is well known to be **NP**-complete. See for example [GJ79].

Given a 3-SAT-formula we construct a labeled complete directed grid, such that there exists a path from the start vertex to the destination vertex, that complies with a fixed (independent of the formula) regular expression, if and only if the formula is satisfiable.

Let $F = (X, C)$ be a 3-CNF formula, $X = \{x_1, x_2, x_3, \ldots, x_n\}$ the set of variables, and $C = \{c_1, c_2, c_3, \ldots, c_m\}$ the set of clauses.

For the beads and holes argument we need $n + 2$ repetitions of the sequence of clauses. To do this, we construct an extended sequence $d_1, \ldots, d_M$ of clauses that consists of exactly $n+2$ copies of the original sequence of clauses ($M = m(n+2)$). It is important that the ordering of the basic sequence remains unchanged between different copies. This extended sequence of clauses is used in the sequel for constructing the graph.

We will describe a grid rotated by 45 degree against the coordinate system that we use to define levels (i.e. rows) and columns. There are three different types of vertices—(i) clause-vertices, (ii) variable-vertices and (iii) join-vertices. The layout of these vertices is illustrated in Figure 4 by means of an example. The coordinate system essentially has for every clause C_i a vertical line with the name of the clause and for every variable x_i a horizontal line named by the variable. These lines are shown as dotted lines in Figure 4. The vertices of the grid lie in the middle of the so formed line-segments. Clause-vertices lie on dotted vertical (clause-) lines and are drawn as rhombuses. Variable vertices lie on horizontal (variable-) lines and are drawn as circles. There are additional bottom and top line of join vertices that are drawn as hexagons. It is easy to see that this grid is part of a complete square grid with $(M + n + 2)^2$ nodes.

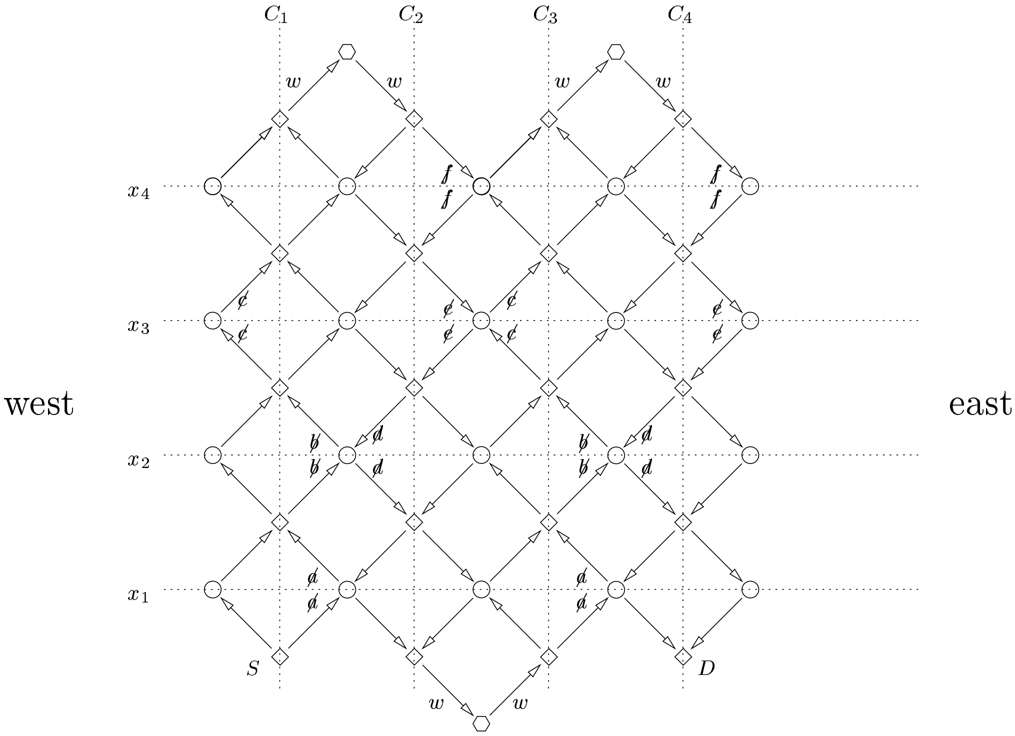

FIG. 4. *Graph corresponding to the formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_4)$, west-true, east-false, in general upward edges are labeled a, b and c, downward edges are labeled d, e and f. The label $\not{a}$ in the figure states that the corresponding edge is not labeled with a (by this deviating from the general rule). For simplicity the additional $4 = (4+2)-2$ repetitions of the basic clause-sequence are omitted.*

To simplify the description we assume that M is even. For the rest of the proof, by a slight abuse of notation, we use the phrase edges incident on a vertex v to mean both the incoming and outgoing edges that have v as one of the endpoints. The start node S is the lowermost clause node on C_1, the destination node D the lowermost clause node on C_M. To achieve paths of the snake like form depicted in Figure 3, we direct all edges incident on a clause-vertex on an odd numbered column upward and all edges incident on a clause-vertex lying on even numbered columns downward.

This orientation is depicted in Figure 4. In order to enforce the overall shape of admissible paths, we label the edges incident on join vertices with w. Furthermore, with important exceptions to be described later, upward edges are in general multi-labeled with a, b and c, where as downward edges are in general multi-labeled with d, e and f. Let us define the regular expression

$$R = \Big((a^* \cup b^* \cup c^*)ww(d^* \cup e^* \cup f^*)ww\Big)^* (a^* \cup b^* \cup c^*)ww(d^* \cup e^* \cup f^*)$$

and the corresponding language $L = L(R)$.

The final construction will have some of the labels removed; this only makes the set of feasible paths smaller. The following proposition summarizes a key property of the above construction.

PROPOSITION 21. *Let P be a simple path in the described graph from S to D complying with the regular language L. Then the following holds: (i) P can be partitioned into a set of legs and (ii) there exist m consecutive legs that have identical shape.*

Proof. **(i)**: Call every subpath labeled with $w(d^+ \cup e^+ \cup f^+)w$ or $w(a^+ \cup b^+ \cup c^+)w$ a leg. Here $x^+ := x^*x$ is the usual shorthand used in regular expressions. As the label of P is in the given regular language, this defines a partitioning of P into M subpaths L_i. Because of the labeling of the grid, all legs have one endpoint at the lower join level and one at the upper join level. Because of this all L_i have the same length. Additionally every second vertex of a leg is a clause-node, and all these clause nodes are in the same column. As we traverse a leg from the low numbered end to the high numbered end, we deviate from the vertical line passing through variable-vertices on the way. The sequence of deviations (east/west) define the *shape* of the leg. The shape of a leg is used to infer an assignment to the variables as discussed later.
(ii) As P is simple, every node of the grid is visited at most once by P. As a result the shapes of two neighboring legs L_i and L_{i+1} are not independent. If L_i deviates at (variable-) level j to the east, L_{i+1} may not deviate to the west on that level. As there are $(M+1)n$ variable-vertices in the grid and Mn variable-vertices on P, there are exactly n variable-vertices not used by P. An averaging argument yields that there must exist m consecutive columns of variable-vertices that are all used by P. This implies that all legs in that range must have identical shape. □

These m legs will be used as multiple copies of an encoding of an assignment of truth values to the variables. The removing of some of the labels of the graph will allow us to enforce the semantics of the clauses. Given the shape of a leg we use the following rule (denoted rule **R**) to infer an assignment to the variables:

> If the shape deviates on level x_i to the west (east), we assign x_i the truth value TRUE (FALSE).

Note that we have m legs with identical shapes and thus we have a consistent assignment to the variables across clauses. The constraining regular expression and appropriate labels to the edges also have to ensure that the following holds. The leg corresponding to each C_i can only have shapes such that the corresponding assignment to the variables using rule **R** makes C_i true. For an odd numbered clause $C_{2k+1} \equiv (u_1 \vee u_2 \vee u_3)$, we remove labels from the graph in the following way: If $u_1 = x_i$ (resp. $u_1 = \neg x_i$) we remove the label a from both the edges incident on the variable-vertex x_i that is to the east (resp. west) of the column C_{2k+1}. For u_2 (resp. u_3) the label b (resp. c) is removed in the same way on the level corresponding to the variable of the literal. For even numbered clauses d, e and f take the role of a, b and c.

PROPOSITION 22. *Let L_k be a leg of a path in G from S to D on column C_k complying with L and let A be the assignment corresponding to the shape of L_k using rule* **R**. *Then the labeling of L_k can be chosen to be of the form $wxx\dots xxw$ with $x \in \{a,b,c,d,e,f\}$ if and only if A evaluates C_k to TRUE.*

Proof. Let A evaluate C_k to TRUE. Then at least one of the literals of C_k is TRUE. We can choose the x accordingly (for example $x = a$ if it is the first literal and k is odd). On the level corresponding to this positive (resp. negative) literal the path deviates to the west (resp. east), and can thus be labeled with x; independent of its shape L_k can be labeled with x on all other levels.

Conversely, if the leg is labeled in $wxx\dots xxw$, we can focus on the literal of the clause corresponding to x (for example the first literal if $x = a$). As the labeling at the level of that variable is only available in the correct direction, A must evaluate the clause to true. □

This completes the construction of the graph. We now prove the correctness of the reduction.

Suppose there exists a satisfying assignment A to the formula. Then we choose the shape of the path from S to D in the snake like fashion with the additional property that we deviate to the west on level x_j if x_j is set to TRUE by A, otherwise we deviate to the east. Since A is a satisfying assignment, each clause contains at least one literal that is set true by A. So we choose the labeling on all legs L_k of the path in column C_k corresponding to this literal of C_k. This yields a simple path that complies with L.

Conversely, let there be a simple path from S to D complying with L. By Proposition 21, we know that the path has m consecutive legs of the same shape corresponding to an assignment A. By construction of the extended sequence of clauses and Proposition 22 we know that A satisfies all of the original clauses. So the formula is satisfiable. □

In fact this results stays valid for a smaller class of infinite regular languages.

DEFINITION 23 ([Str94]). *A language L is called locally testable, if there exists a k and a finite set S such that*

$$w \in L \iff (\forall v <_k w : v \in S).$$

Here $v <_k w$ stands for the fact that v is a subword of length k of w.

COROLLARY 24. *The* REG-SIP *Problem is* **NP**-*hard for a complete multi-labeled directed grid and a fixed locally testable language.*

Proof. In the proof of Theorem 20 the constraining regular language can be replaced by the regular language L defined as follows:

$$L = \left\{ v \in \Sigma^* \;\middle|\; x_1 x_2 <_2 v \to \Big(x_1 \in \{a,b,c,d,e,f\} \to x_2 = x_1 \lor x_2 = w\Big) \right\}$$

This condition is equivalent to stating that any sequence of $y \in \{a,b,c,d,e,f\}$ may only be ended by a w. Starting from the first symbol in the word, L guarantees that this single symbol is repeated until the next w. The labeling of the grid ensures that this w is at the top join vertex between the columns C_1 and C_2. There the simplicity enforces another w and it follows inductively that L enforces the snake like shape of the path as well as the uniform labeling of the legs. So it is as strong as the language used in the proof of Theorem 20.

It should be noted that Lemma 16 cannot be applied immediately in this situation. Nevertheless the result of the corollary extends. This is easy to verify using a slight modification of the graph proposed in the proof of the Lemma 16 and a modification of the language. This makes use of the fact that the directionality can readily be omitted and that constructions of the form $x_i = y \rightarrow x_{i+5} = y$ are also expressible in locally testable languages. ☐

Note that the result of Theorem 20 immediately extends to other graph classes:

COROLLARY 25. *Let $\mathcal{C}$ be a class of graphs such that for all $k > 0$ there is an instance $\mathcal{I} \in \mathcal{C}$ that contains as a subgraph a $k \times k$-mesh and both the graph and the subgraph are computable in time polynomial in k, then the* REG-SIP *problem is* **NP***-hard for $\mathcal{C}$.*

Thus the REG-SIP *problem is* **NP***-hard even for (i) complete cliques, (ii) interval graphs, (iii) chordal graphs, (iv) complete meshes, (v) complete hypercubes, (vi) permutation graphs.*

6.3. Hardness of CFG-SIP**.** We show that CFG-SIP, the problem of finding a simple path complying with a context free language, is **NP**-hard even on graphs with bounded treewidth. Before formally stating the proof, we give the overall idea. We present a reduction from 3-SAT (See e.g. [GJ79] for definition). The basic idea is to have a path consisting of two subpaths. The first subpath uniquely chooses an assignment and creates several identical copies of it. The second subpath checks one clause at every copy of the assignment and can only reach the destination if the assignment satisfies the given formula.

Consider the language $L = \{w\#w^R\$w\#w^R \dots w\#w^R | w \in \Sigma^*\}$. As is standard, w^R denotes the reverse of string w. At the heart of our reduction is the crucial observation that L can be expressed as the intersection of two context free languages L_1 and L_2. Consider $L_1 = \{\, w_0\#w_1\$w_1^R\#w_2\$w_2^R t \dots w_k\$w_k^R\#w_{k+1} \mid w_i \in \Sigma^* \,\}$ and $L_2 = \{\, v_1\#v_1^R\$v_2\#v_2^R \dots v_k\#v_k^R \mid v_i \in \Sigma^* \,\}$. To see that $L = L_1 \cap L_2$, observe that $w_0 = v_1$, $v_1^R = w_1$, and for all i, $w_i^R = v_{i+1}, v_{i+1}^R = w_{i+1}$, establishing that $v_i = v_{i+1}$ holds for all i, and thus $L = L_1 \cap L_2$.

Now, imagine w representing an assignment to the variables of a 3-CNF formula with a fixed ordering of the variables. For every clause of the formula, we create a copy of this assignment. L is used to ensure that all these copies are consistent, i.e. identical.

Note that we have two basic objects for performing the reduction—a CFG and a labeled graph. We will specify L_1 as a CFG and use the labeled graph and simple paths through the graph to implicitly simulate L_2. Recall that there is a straightforward deterministic pushdown automaton M for accepting L_2. Our graph will consist of an "upward chain" of vertices and a "downward chain" of vertices along with a few additional vertices. The upward chain will simulate the behavior of M when it pushes w on the stack. The "downward chain" will then simulate popping the contents of the stack and verifying that they match w^R. We will call such a gadget a "tower" as an analogy to the stack of M.

We now describe the proof in detail. For the purposes of simplicity, we prove the results for directed graphs, the extension to undirected graphs follows with Lemma 16.

THEOREM 26. *The* CFG-SIP *problem is* **NP***-hard, even for Graphs of constant treewidth and a fixed deterministic Context Free Language.*

Proof. Reduction from 3-SAT. Let $F(X, C)$ be a 3-CNF formula, where $X = \{x_1, \dots, x_n\}$ denotes the set of variables and $C = \{c_1, \dots, c_m\}$ denotes the set of clauses. Corresponding to F, we create an instance $G(V, E_1 \cup E_2)$ of CFG-SIP as follows. We will describe the reduction in two parts—first the subgraph (V, E_1) and then the subgraph (V, E_2).

The subgraph (V, E_1) is constructed as follows: Corresponding to each clause c_j, we have a Tower T^j. It consists of n "simple-path stack-cell" gadgets H, for each variable one. This basic gadget is depicted in Figure 5.

Consider a simple path p from one of the bottom nodes (marked by a square in the Figure) to one of the top nodes. Because of the labels of this gadget, we can define the *signature* y of this path by $l(p) = cyc$ with $y \in \{a, b\}$. Let q be another simple path, that has no vertex in common with p, starts at one of the top nodes and ends in one of the bottom nodes. Then we can again properly define the signature z of this path by $l(q) = czc$.

Because of the nodes α and β, the signatures are identical, i.e. $y = z$. Furthermore, if p uses the node x_i', the node x_i is not used at all, and q has to use the node $\neg x_i$. Similarly, if p uses the node $\neg x_i'$, the node $\neg x_i$ is not used at all, and q has to use the node x_i.

These gadgets are now composed to form towers T^j, by identifying the top terminal nodes of H_i^j with the bottom terminal nodes of H_{i+1}^j. The tower has four levels corresponding to every variable. We call this a *floor* of the tower. The bottom of the tower T^j is connected to the bottom of the tower T^{j+1}. The start vertex is connected to the bottom of the tower T^1. These connections are depicted in Figure 6. Before we describe the remaining edges, we discuss the properties of a tower.

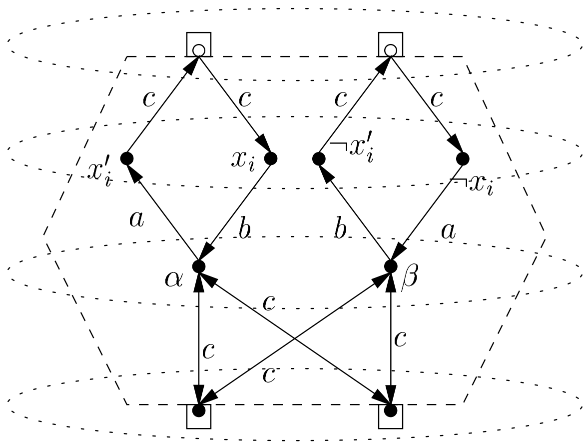

FIG. 5. *the gadget H_i used to implement the tower, forcing the assignment to be spread consistently over the graph. Dashed ellipses denote the so called level sets.*

Consider a Tower T^j and a simple path r labeled according to $(c(a \cup b)c)^* \#(c(a \cup b)c)^*$ that starts at one bottom vertex and reaches the other bottom vertex. Such a path has the following important properties:

1. The path r consists of two simple subpaths p and q separated by an edge labeled with $\#$. p starts at the bottom of the tower and is labeled according to the regular expression $(c(a \cup b)c)^*$. On every floor it uses exactly one of the nodes $\{x_i', \neg x_i'\}$, by this realizing an assignment to the variables of X. This assignment uniquely corresponds to the labeling of this path.

2. The constraints of *simplicity* and the direction of edges implies that q has the following structure: it starts at the unused top vertex, is labeled with $l(q) = l(p)^R$, avoids already used nodes, and reaches the unvisited bottom vertex. Furthermore q is *uniquely* determined given p.

3. Note that for each variable x_i, the simple path r visits exactly one of the nodes in $\{x_i, \neg x_i\}$. If the path reflects an assignment in which x_i is true, then x_i is unused and $\neg x_i$ is used and vice versa. These free nodes will be used in the second part of the reduction to verify the chosen assignment is indeed satisfying F.

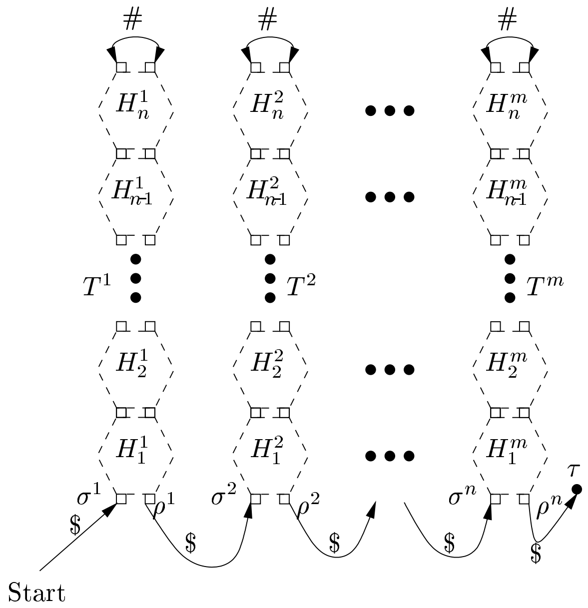

FIG. 6. *Assembling the gadgets, building the graph G_1.*

We now assemble the towers to form the graph G_1 depicted in Figure 6.

PROPOSITION 27. *Any simple path in G_1 starting at the start node s, and reaching the intermediate node τ, with the constraint that the labeling belongs to $\$((c(a \cup b)c)^*\#(c(a \cup b)c)^*\$)^*$ generates the language*

$$L_2 = \{\, \$w_1\#w_1^R\$w_2\#w_2^R\$ \ldots w_n\#w_n^R\$ \mid w_i \in (c(a \cup b)c)^* \,\}.$$

Proof. The statement follows from the above.□

We now choose the constraining CFL for the path from s to τ to be
$L_1 = \{\, \$w_1\#w_2\$w_2^R\#w_3\$w_3^R\# \ldots w_{k-1}\$w_{k-1}^R\#w_k\$ \mid k \in \mathbb{N}, w_i \in (c(a \cup b)c)^* \,\}$.
The following important lemma follows from Proposition 27 and the definition of L_1.

LEMMA 28.
1. *Proposition 27 enforces that in every tower the used and unused nodes can be uniquely interpreted as an assignment to the variables of X.*
2. *L_1 enforces that these assignments are consistent across two consecutive towers.*

We now describe the subgraph (V, E_2). The label of each edge in this subgraph is u. The subgraph is composed of m subgraphs $D_1 \ldots D_m$, the subgraph D_i corresponding to clause c_i, depicted in Figure 7. Every D_i basically consists of four simple chains. The first one goes up the tower. There it splits into three downward directed paths. Each of them corresponds to a literal of c_i. The node in the tower T_i corresponding to that literal is used as part of the path. At the very bottom the three paths are joined and connected to D_{i-1}. At the boundaries D_0 is replaced by d and D_{m+1} by τ. This completes the description of the graph $G(V, E_1 \cup E_2)$.

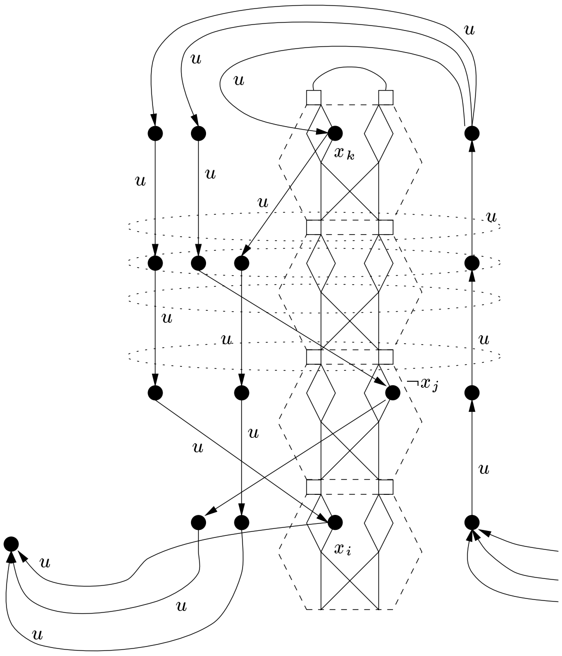

FIG. 7. *One tower of the second part of the graph, according to the clause* $(x_i \vee \neg x_j \vee x_k)$.

The instance of CFG-SIP consists of the graph $G(V, E_1 \cup E_2)$ and the constraining CFL $L = L_1 \cdot u^*$. This enforces the path to go through every tower using edges in $G(V, E_1)$, visit vertex τ and then use the edges of $G(V, E_2)$ to reach d.

We now prove the correctness of the reduction. Suppose, there exists a satisfying assignment for F. Then we choose the path from s through all the towers to τ accordingly. For the remaining path to d we use the fact that for every clause of F at least one literal is true. Choosing the downward directed subpath according to this literal we can complete a simple path to d. By construction, the label of this path complies with the context free grammar.

Conversely suppose there exists a simple path in G from s to d satisfying the labeling constraint. The alternation of # and \$ in L enforces that such a path visits every tower, then the vertex τ and finally the destination. In this situation we can define an assignment A according to the path in the first tower. By Lemma 28, and the path being simple, we know that A is consistently represented in all the towers. The second part of the path shows that A is a satisfying assignment for F.

Finally it is easy to verify that G has a constant treewidth. For this we describe a tree decomposition of the graph. The up to four nodes of one level in a gadget H_i and the up to four nodes of the path D_i form so called level-sets. Two neighboring level-sets together, and two bottom level sets of neighboring towers respectively, form a set of the tree decomposition. Additional sets for s, d, and τ with all the adjacent nodes are needed. This shows that the treewidth is bounded by 16. ☐

Actually the result of Theorem 26 can be extended to obtain the following:

COROLLARY 29. CFG-SIP *is* **NP**-*hard when restricted to graphs of treewidth 3 and a fixed* linear, deterministic *CFG.*

Proof. The proof is similar to the proof of Theorem 26 and thus we only describe the necessary modifications needed to prove the corollary. First we can replace the second part of the graph, used to verify that the assignment satisfies the formula. This is done as in the proof of Theorem 20, by replacing a (and b) by three symbols a_1, a_2 and a_3 (b_1, b_2, b_3). We modify the language such that for the context-free part all a_i are equivalent and all b_i are equivalent. Then we add a regular component to the language (take the intersection), enforcing that on one leg all the indices have to be identical. By removing certain labels we can make sure that in every tower one clause is tested.

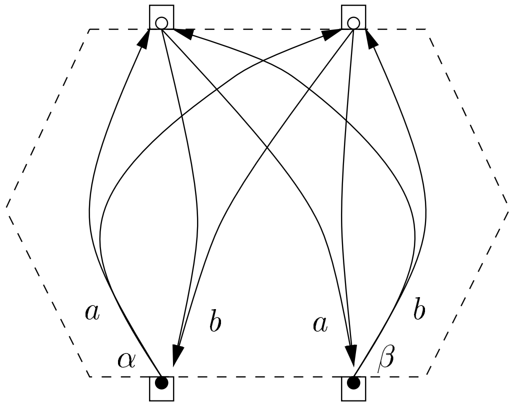

FIG. 8. *the smaller gadgets replacing the H_i used in the original construction. a stands for a_1, a_2, a_3, b for b_1, b_2, b_3. To encode the clauses some of these labels get removed, like in the proof of Theorem 20.*

Since we do not need the free nodes anymore, the gadget H_i depicted in Figure 5 can be replaced by a much smaller one, depicted in Figure 8. Note that replacing parallel directed links by disjoint paths of length two does not increase the treewidth in this situation. The additional nodes and edges can be covered in a tree decomposition by some more leaf-sets of size three. This does not increase the width of the tree decomposition. In total the treewidth of the modified construction is reduced to 3.

Taking a new alphabet symbol x we define the language $H = \{ wxw^P\#v | w \in L_1, v \in (\Sigma - \{\#\})^* \}$. It is easy to see that H can be specified by a linear context free grammar if L_1 is a regular language. Here w^P is defined to be the reverse of w with bottom marker \$ and top marker # exchanged. We take L_1 to be a regular language that enforces the path to use the top marker of every tower. It is sufficient to argue that the proof of Theorem 26 can be modified such that the constraining context free language can be replaced by H.

For this we double the sequence of variables, introducing a new set of variables $\{x'_i = x_{2n-i+1}\}$ yielding the extended sequence $x_1, x_2, \ldots, x_n, x'_n, x'_{n-1}, \ldots, x'_1$. The new variables are only used while constructing the towers, but not for the evaluation of clauses. Additionally we have to incorporate the middle marker x into the graph. Assuming the number of clauses m is odd, this means, that we modify the tower $m/2$ at the level of the lowermost gadget $H_{n+1}^{m/2}$ created for the new variable x'_n. In this gadget we replace both upward edges by paths of length two. The upper edge gets the original label, the lower edge gets labeled with x. The proof of correctness consists of showing that the new H ensures a single assignment is represented by all the subpaths in the towers. This can be shown inductively from the middle symbol x outwards. The way the information is spread is depicted in Figure 9. The induction

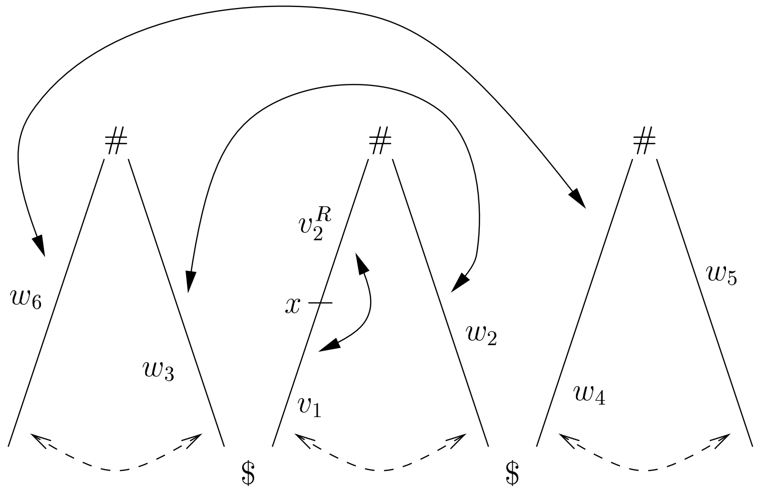

FIG. 9. *schematic labeling of the path;* $w_1 = v_1 v_2^R$*; solid arrows stand for correspondence enforced by the language, dashed arrows for correspondence resulting from the graph.*

starts with the fact that the signature w_1 of the upward path in tower $m/2$ is forced by H to comply with $v_1 = v_2^R$, leading to the definition $w_1 = v_1 v_2^R$, with the property $w_1 = w_1^R$. So we do not have to distinguish between w_1 and w_1^R. Then the graph enforces $w_2 = w_1$, H enforces $w_3 = w_2$ and so on. This establishes that all the (lower halfs of the) signatures are identical. The remaining part of the proof follows the details given for Theorem 26 closely. □

Note that the language H in the proof of Theorem 26 can also be accepted by a deterministic log-space bounded Turing machine having a two-way input tape.

6.4. Algorithm for REG-SIP **on graphs of bounded treewidth.** In contrast to the above **NP**-hardness results we show that for graphs of bounded treewidth the problem REG-SIP is solvable in polynomial time. The class of treewidth bounded graphs includes interval graphs and Chordal graphs with *bounded clique size*, complete meshes, with fixed length or width, outer planar graphs, series parallel graphs, etc. This easiness result along with the hardness result in the preceding section also imply that these results are in a sense the best possible. We will use the notion of a nice tree decomposition discussed in [Bo92].

DEFINITION 30. *A tree decomposition* $\langle \{ X_i \mid i \in I \}, T = (I, F) \rangle$ *is* nice, *if one can choose a root r in the tree such that:*

- T *is a binary tree,*
- *if* i *is a leaf, then* $|X_i| = 1$ *(Start node);*
- *if* i *has two children* j *and* k *then* $X_i = X_j = X_k$ *(Join node);*
- *if* i *has one child* j*, then there exists a vertex* v *such that either*
  - $X_i = X_j \setminus \{v\}$ *(Forget node), or*
  - $X_i = X_j \cup \{v\}$ *(Introduce node).*

We use the fact, that there exists always a nice tree decomposition of optimal width (treewidth), and that it can be constructed in linear time [Bo92].

THEOREM 31. *The* REG-SIP *problem is solvable in polynomial time on treewidth bounded graphs*

Proof sketch: Given a treewidth bounded graph G along with a nice tree decomposition $T(I, F)$, we describe an algorithm, that computes tables of (partial) shortest simple paths in a bottom up fashion in T. Specifically, we have one table for every set X_i corresponding to the node i of T. We describe the entries of these tables and how to compute the values (i) for leaf sets and (ii) for internal nodes from the values in the tables of the child node(s). Since the number of values computed as well as the number of tables is polynomial in the size of G, this yields a polynomial time algorithm.

The complicated task is to keep track of is the simplicity of the path or more precisely the nodes used by the partial solutions represented by entries of the table. We have an entry for each type of path going through the set of nodes the table is attached to. Since these sets are separators in G it is not necessary to keep the complete information about possible paths "behind" the separator. For the remainder of this section, we use k to denote the treewidth of the tree decomposition T. In the following, given for a set X_j corresponding to a node i in T we give a characterization of the distinct subsolutions that need to be maintained.

The indices (also called "atoms" for this proof) used to describe the subsolutions stored in the table are the following:

1. Node in S the path starts,
2. State the automaton is in before reading in the label of the path,
3. Node in S the path ends,
4. State of the automaton after having read in the label of the path.

We accept the special situation of one node paths, i.e. indices with identical nodes and states.[2] There are two special "half" atoms standing for a beginning segment and an end segment of the path: There we have a node and a state, identifying the end node of a simple path that starts at the source, such that its label can lead the automaton to the specified state. Similarly, for the end segment of a path we have a node and a state such that the labeling along the path gets the automaton from this state to an accepting state at the sink. Thus the subsolutions are completely described by

> the $k/2-1$ atoms plus the two special half atoms together with a set of other used nodes in X_i

We only need to allow the atoms to be empty; meaning that they do not denote any path. Noting that the total number of ways to partition a set of size k is $O(k^k)$ leads to an upper bound of $O((k \cdot |NFA|+1)^k \cdot 2^k)$ for the number of entries in the tables[3] which is polynomial in the size of the NFA for fixed treewidth k.

Note that the entry in the table where both of the special half atoms are empty stands for a complete solution in the already visited part of the graph. For every type of partial solutions we maintain the total length of the shortest partial solution.

We now describe the tables more formally. For the leaf sets consisting of a single node v the table is easy to compute. It has entries with value 0 for the following partial solutions:

$s \neq v \neq d$ for all states i: the length 0 path v-v, with start and end states i, no additional nodes used.

$v = s$ the length 0-path from s to v ending in the initial state.

$v = d$ for all accepting states $f \in F$: the length 0-path from v to d beginning in state f.

In order to compute the tables in a bottom up fashion in T, we need to consider

[2]Here we could actually account for ε moves in the automaton if we choose to.

[3]We have not attempted to optimize the size of the tables. To do so one could understand a certain usage pattern that chooses one of the $(k/2)!$ possible representations.

three possible cases depending on the type of nodes in T. Let i be a node in T.

i **is a Join Node:** 1. Letting j and l be the children of i, we know that $X_i = X_j = X_l$. Combine a partial solution stored in the table associated j with a partial solution stored in the table associated with l. For all pairs of types of partial solutions check if they can be combined to form another partial solution (no commonly used nodes, matching boundary node and state of the partial solution, respecting special cases for source and destination subpaths). Create the new type and compute the value of the combined solution. If the type does not yet exist in the table or the newly computed value is smaller than the old table entry, update the table entry. This is justified by the observation that the described subsolutions associated with j and k are disjoint except at the set of separator nodes X_i.

2. Keep the componentwise min of the tables: for all types of partial solutions (their description match as the two sets of the decomposition are identical), keep the smaller value. If the solution according to a type does not exist, we assume its value to be infinity. (This can also be seen as (1.), where we combine the entry with an "empty sub solution" of cost 0 from the left and from the right and then choose the better one.)

i **is a Forget node:** Let v be the node removed from the set in the nice tree decomposition. We discard all partial solutions that contain a path with endpoint v. In all partial solutions, we delete v in the set of used nodes. For the resulting identical subsolutions we keep the one with minimum value.

i **is an Introduce node:** Let v be the new node, e_i the edges between v and nodes in the set of the child.

1. Set up the new node. Copy all known partial solutions. Create new partial solutions by combining all known with all solutions created according to the rules stated above for leaf sets of the form $\{v\}$.

2. Include the incident edges one by one. Consider all possible new paths using this edge.

The correctness of the algorithm follows by noting the following:

1. Given an entry in the root of the table for the existence of a path p of a given length, we can easily find such a path recursively from subsolutions associated with the children of the root.

2. Conversely, let p be one of the optimal (shortest) solutions. Let X_r be the set associated with the root of T. Assume that r is a join node and let l and w be the left and right children. The argument for the other two cases is similar and thus omitted. It is easy to see that the path p can be broken into two paths p_1 and p_2 (p_2 could be empty) such that p_1 is a sub solution maintained at l and p_2 is sub solution maintained at w. □

The following generalization of the previous result holds:

COROLLARY 32. *Let L be a fixed language decidable by a one-way input tape,* log*-space bounded nondeterministic Turing machine. Then the problem of finding a shortest simple path from source to destination according to L is solvable in polynomial time on treewidth bounded graphs.*

Proof. The length of a simple path in the graph is bounded by n. The number of configurations of a nondeterministic Turing machine using $\log n$ tape cells is polynomial in n. No other configurations of the machine can be used while recognizing a word of length smaller or equal n. As the machine reads the input tape in one direction only, we can create an NFA with states representing configuration that (i) decides L for words of length $\leq n$ and (ii) has a size polynomial in n. Using the algorithm in the proof of Theorem 31 with NFA M, we can compute the sought path in polynomial time. □

Note that $a^n b^n c^n$ and all languages accepted by pushdown automata having only one stack-symbol are examples of such languages.

6.5. Algorithm for Acyclic Graphs. Another well know situation in which shortest simple paths are feasible is for acyclic graphs. This stems from the fact that all paths in an acyclic graph are simple and by this the shortest and the shortest simple path always coincide. This together with the results of Section 5 yields the following result:

COROLLARY 33. *The problem* CFG-SIP *is solvable in polynomial time on directed acyclic graphs.*

Making use of the fact that the length of a simple path is bounded by the size of the graph, we get the following:

COROLLARY 34. *The problem of finding a shortest simple simple from source to destination in an directed acyclic graph according to a formal language L is solvable in polynomial time if there exists a polynomial time computable CFL R of size polynomial in n with the following property:*

$$x \in \Sigma^*,\ |x| \leq n :\ \ (x \in L \leftrightarrow x \in L(R))$$

7. Extensions and Applications. In this section, we briefly discuss extensions and applications of our results to problems in transportation science. For many of these applications, it is possible to devise dynamic programming based methods to solve the problems; our aim is to convey the applicability of the general methodology proposed here.

7.1. Node Labels and Trip Chaining. Consider the problem, where instead of the edges the nodes have labels, and the constraint is on the compound node label of a path. Easy transformations of the input show, that all the results we develop for the edge labeled case are also true for the node labeled case. We can transform this kind of instance into an edge labeled one by the following steps: The network stays the same, the edges get labeled with a new symbol. Every node gets an additional loop attached. This loop gets the label(s) of the node. Then the language has to be extended such that (i) exactly every second symbol has to be the new edge symbol and (ii) the word without the edge symbols is in the original language. It is easy to see that regular and context free languages are closed under this operation. This shows, that easiness results for edge label constraints imply easiness results for node label constraints. If we have an edge labeled graph, we split the edges and insert a node with the edge label, and label all the old nodes with one new symbol. The language has to be extended like above. This construction transfer edge label constrained hardness results to node label constrained results.

The node label extensions turn out to be useful in modeling transportation related problems. For example, in many transportation applications (e.g. TRANSIMS) one needs to find paths for travelers, that need to visit a fixed sequence of location types. For instance, we might want to find a shortest path from home to home that visits some locations e.g. ATM–gas station–supermarket–post office in this particular order, but with the freedom to choose one of the several ATM's (gas stations,...) in the city. The problems of this type are referred to in the transportation literature as *trip chaining* problems. The problem cannot be solved by direct application of Dijkstra's shortest path algorithm to find best paths between two consecutive subdestinations and concatenating these paths. By treating each destination type as a node label and constructing a simple regular expression, we can select places for the subdestinations and find shortest path in networks that satisfy the precedence constraints using the polynomial result discussed in this paper.

7.2. Finding Alternatives to Shortest Paths. There has been considerable interest in algorithms for variations of shortest paths[AMO93]. For example, in a recent paper, by Scott, Pabon-Jimenez and Bernstein [SJB97], the authors consider the following problem — given a graph G, a shortest path SP in G and an integer parameter, find the shortest path $SP1$ in G that has at most k links in common with SP. Call this the BEST k-SIMILAR PATH. In [SJB97], the authors present an integer linear program formulation of the problem and present a heuristics based on Lagrangian relaxation. It can be verified that the heuristic takes exponential time in the worst case. Quoting [SJB97], *"The link overlap constraint thus makes finding the best path much more difficult (i.e. shortest path problems with a single constraint are* **NP***-hard)"* thus suggesting that BEST k-SIMILAR PATH is likely to be **NP**-hard. Here we show that this problem is solvable in time $T(k|G|)$ where $T(n)$ denotes the time taken to find a shortest path on a graph with n vertices. This substantially improves on the exponential time algorithm given in [SJB97]. Our approach for solving the problem is based on using our algorithm for regular expression based shortest paths. The approach uses the fact that given a k, and symbol $a \in \Sigma$, the language consisting of all words $w \in \Sigma^*$ with no more than k occurrences of the symbol a in them, is regular. Given a graph G, a shortest path SP in G and an integer parameter k, we perform the following steps:

1. Label the edges on the shortest paths by a and all the other edges in G by b.
2. Construct an NFA M that accepts all strings that have no more than k occurrences of a. The corresponding automaton M is shown in Figure 10 (and happens to be deterministic).
3. Find a shortest path in G with the constraint that its label is in $L(M)$.

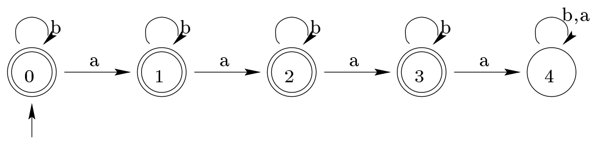

FIG. 10. *Automaton used to count shared links, accepting if less than 4*

The proof of correctness is straightforward. We note that in this particular case, we in fact always get a simple path, since we can always remove a loop from the path without increasing its length or acceptability of the shorter string.

7.3. Handling Left Turn and U-Turn Penalties. Suppose, we are told that in our road network there is additional cost incurred when we take certain left turns. This is a common scenario in transportation science and is referred to as turn penalties. This is not an unlikely scenario. In fact the current Dallas FT Worth Case study has exactly this situation. Specifically, several left turns in the study area are prohibited which amounts to saying that the cost of taking that edge if it is a left turn is ∞. Moreover, the infrastructure change that is proposed for the case study intends to make a series of left turns illegal near the area of Valley View Mall and the Galleria.

A well known reduction from the original problem to the problem of finding a shortest path in a modified network (see [AMO93] for details) can be used to solve this problem. (The basic idea is to replace each intersection by a clique of size 4. A slightly more complicated subgraph is required for directed graphs.) Instead of giving a penalty to a turn, suppose we wish to find a path in which we do not take more than say k-left turns. This variant of the problem cannot be solved by a direct application of Dijkstra's algorithm, but is amenable to an efficient solution using the formal language approach. To do this we again replace each intersection by a clique of size four and then add appropriate labels to each of these edges. We then construct an automaton, which accepts strings that contain at most k-labels corresponding to left turns. This can be constructed along the lines of k-similar path problem. The rest of the details are straightforward. Now consider a more complicated query in which we wish to find a path such that the number of left turns are a small fraction of the total number of edges. It is easy to see that this can be written as a context free grammar and thus again can be solved efficiently.

7.4. Time dependent Networks and Multi-Criteria Shortest Paths. In several transportation applications, it is desirable to solve shortest path problems on networks in which the edge weights are a function of time. In [OR90] Orda and Rom consider this type of problem for various waiting policies and function classes. One of their basic algorithms is a dynamic programming on functions. Combining this with our results, we obtain a polynomial time algorithm for CFL-constrained shortest paths in time depending networks.

As a final application, consider bicriteria and in general multi-criteria shortest path problems. For instance, we might have two different weight functions on each edge a function $c(e)$ that captures the cost of using that edge and a function $t(e)$ that captures the time it takes to traverse the edge. The aim of the bicriteria shortest path problem aims at finding a minimum cost path from a source s to a destination d, that obeys a given budget bound B on the time taken to go from s to d. This problem has been studied extensively in the literature (see [MRS+95] and the references therein). Given that the cross product construction simply constructs multiple copies of the basic graph, it is easy to design polynomial time approximation scheme for the bicriteria shortest path problem subject to labeling constraints. The idea is a straightforward combination of the ones outlined here and those used for designing approximation scheme for the basic bicriteria problem.

7.5. Other Formal Languages. As expected, attempts to extend the polynomial time algorithms to more general grammars such as the context sensitive grammars fail to yield polynomial time results. Intuitively, the hardness of the problems is due to the fact that *emptiness* and *recognition problems* for context sensitive grammars are undecidable and **PSPACE**-complete respectively.

Consider the problem for context sensitive grammars. It is easy to show that the problem (CSG-ShP) is undecidable even for a fixed log-space decidable context sensitive language. It is easy to see, that the language

$$L = \left\{ w\#^i \mid w \in \{0,1\}^*,\ \mathrm{TM}(w) \text{ halts on an empty input using space } \log i \right\}$$

is context sensitive. It is in fact log-space computable. It is also straightforward to see that

$$L' = \{w \mid \exists i\ w\#^i \in L\}$$

is another way of stating the halting problem and is thus undecidable.

Showing that we can use CSG-ShP to decide L' establishes that CSG-ShP is itself undecidable. For $w \in \{0,1\}^*$ we construct a directed chain labeled according to w with start s and end d. We additionally put a loop from d to d labeled with #. The fixed constraining language is L. Now $w \in L'$ is equivalent to the existence of a (in general not simple) path from s to d in this graph. Note that this graph is nearly a tree.

In contrast, the CSG-SiP is **PSPACE**-complete. Membership in **PSPACE** follows by observing that a simple path in a graph $G(V,E)$ can have at most $|V|$ nodes. Thus a space bounded NDTM can guess a simple path p and then verify that $l(p) \in L(M)$, where M is the CSG. The hardness is shown by reducing the problem of deciding if $w \in L(M)$ to finding a simple path in a directed chain that is labeled according to w. Let s and t denote the two end points of this chain. Then there is a path from s to t whose labels are in $L(R)$ if and only if $w \in L(R)$. Similar extensions hold for other formal languages. Note that for this argument the context sensitive language can be fixed to represent an arbitrary **PSPACE**-complete problem.

8. Conclusions. In this paper, we have presented a general approach for modeling and solving a number of problems that seek to find paths subject to certain labeling constraints. The model was shown to be particularly useful in understanding and solving transportation science problems. The results in this paper provide a fairly tight characterization of the complexity of these problems, varying the type of considered path, the underlying grammar and allowed graph classes. For a number of non-trivial cases this completely characterizes the boundary between easy and hard cases. Our results can also be seen investigating tradeoffs between (i) economy of descriptions of languages used to describe labeling constraints and the (ii) efficient solvability of the corresponding problems.

In [BK+99, JBM99, JMN98], we have obtained a number of additional theoretical as well as empirical results on related topics. Specifically in [JBM99], we show how to apply our results on finding REG-ShP in time dependent networks can be used to solve a number of additional problems arising in transportation science. In [BK+99, JMN98], we have carried out a detailed experimental analysis to validate the suitability of extensions of the algorithm suggested here on realistic transportation networks. We refer the reader to the web-site `http://transims.tsasa.lanl.gov` for comprehensive information about TRANSIMS and related documents.

The results presented here raise a number of questions for further investigation. First, it would be of interest to characterize the class of fixed regular (context free) languages for which the regular expression constrained simple path problems are solvable in polynomial time. It would also be of interest to provide natural formulation of other label constrained subgraph problems. For example, what is a natural way to specify labeling constraints for spanning trees of a graph? A number of possible ways present themselves; the aim is find ways that are both natural and useful in modeling practical problems.

Acknowledgments: We want to thank Harry Hunt III, Sven Krumke, Alberto Mendelzon, S. S. Ravi, R. Ravi, Dan Rosenkrantz, M. Yannakakis, Hans-Christoph Wirth, Stephen Eubank, Kai Nagel, Terence Kelly, Robert White and Brian Bush for several useful discussions, pointers to related literature and suggested improvements in the course of writing this paper. We also thank the anonymous referee for a careful

reading of the manuscript and a number of helpful comments on the earlier draft.

REFERENCES

[AV99] S. Abiteboul and V. Vianu, *Regular path queries with constraints*, J. Computer and System Sciences 58, pp. 428–452 (1999).

[BaL] M. Ben-Akiva and S.R. Lerman, *Discrete Choice Analysis*, MIT Press Series in Transportation Studies, Cambridge, MA, 1985.

[AMO93] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[AC+93] S. Arnborg, B. Courcelle, A. Proskurowski and D. Seese, *An algebraic theory of graph reductions*, Journal of the ACM (JACM), 40:5 (1993), pp. 1134–1164.

[AMM97] G. Arocena, A. Mendelzon, G. Mihaila, *Applications of a Web Query Language*, Proc.6th Int'l. WWW Conf., Santa Clara, April 1997.

[AHU] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, Reading MA., 1974.

[AB+97] D. Anson, C. Barrett, D. Kubicek, M. Marathe, K. Nagel, M. Rickert and M. Stein, *Engineering the route planner for dallas case study*, Technical Report, Los Alamos National Laboratory, February 97.

[AL+91] S. Arnborg, J. Lagergren and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.

[Bo88] H.L. Bodlaender, *Dynamic programming on graphs of bounded treewidth*, Proc. 15th International Colloquium on Automata Language and Programming (ICALP), LNCS 317, pp. 105–118 (1988).

[Bo92] H. L. Bodlaender, *A tourist guide through treewidth*, Technical Report RUU-CS-92-12, Utrecht University (1992).

[BL+87] M. W. Bern, E. L. Lawler and A. L. Wong, *Linear-time computation of optimal subgraphs of decomposable graphs*, J. Algorithms, 8 (1987), pp. 216–235.

[BAL97] V. Blue, J. Adler and G. List, *Real-time multiple objective path search for in-vehicle route guidance systems*, Proc. 76th Annual Meeting of The Transportation Research Board, Washington, D.C. Paper No. 970944, January 1997.

[BKV91] A. Buchsbaum, P. Kanellakis and J. Vitter, *A data structure for arc insertion and regular path finding*, Proc. 1st ACM-SIAM Symposium on Discrete Algorithms (SODA), 1990, pp. 22–31.

[BK+99] C. Barrett et. al. *Routing problems in TRANSIMS: theory, practice and validation*, Technical Report, Los Alamos National Laboratory, March 99.

[CMW87] M. Cruz, A. Mendelzon and P. Wood, *A graphical query language supporting recursion,* Proc. 9th ACM SIGMOD Conference on Management of Data, San Francisco, CA, 1990, 1987, pp. 323–330.

[CMW88] I. Cruz, A. Mendelzon and P. Wood, *G^+: Recursive queries without recursion*, Proc. 2nd International Conference on Expert Data Base Systems, Tysons Corner, CA, 1988, pp. 25–27.

[CLR] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Co., 1990.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco (1979).

[Ha88] F. G. Halasz, *Reflections on notecards: seven issues for the next generation of hypermedia systems*, Communications of the ACM, 31(7), 1988, pp. 836–852.

[HNR68] P. Hart, N. Nilsson, and B. Raphel, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Trans. on System Science and Cybernetics, (4), 2, July 1968, pp. 100–107.

[Ha92] R. Hassin, *Approximation schemes for the restricted shortest path problem*, Mathematics of Operations Research, 17, 1 (1992), 36–42.

[HM95] Highway Research Board, *Highway Capacity Manual*, Special Report 209, National Research Council, Washington, D.C. 1994.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison Wesley, Reading MA., 1979.

[Hu97] U. Huckenbeck, *Extremal paths in graphs*, Foundations, search strategies, and related topics, Akademie Verlag, Berlin, 1997, ISBN 3-05-501658-0,

[JBM99] R. Jacob, C. Barrett and M. Marathe, *Models and efficient algorithms for class of routing problems in time dependent and labeled networks*, Technical Report, Los Alamos

National Laboratory, March 99.

[JMN98] R. Jacob, M. Marathe, and K. Nagel, *A Computational Study of Routing Algorithms for Realistic Transportation Networks*, invited paper accepted subject to revisions in ACM J. Experimental Algorithmics, December 1998. Preliminary version appeared in Proc. 2nd Workshop on Algorithmic Engineering, Saarbrucken, Germany, August 1998.

[Ku77] D.E. Knuth, *A generalization of dijkstra's algorithm*, Information Proc. Lett., 6(1) (1977), pp 1–5.

[HRS76] H. B. Hunt III, D. Rosenkrantz and T. Szymanski, *On the equivalence, containment and covering problems for regular and context free grammars*, J. Computer and System Sciences, 12 (1976), pp. 222–268.

[MW95] A. Mendelzon and P. Wood, *Finding regular simple paths in graph databases*, SIAM J. Computing, 24-6 (1995), pp. 1235–1258.

[MRS+95] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, *Bicriteria network design problems*, in J. Algorithms, 28(1), pp. 142-171, July 1998.

[Pa94] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Massachusetts, 1994.

[Ro88] J. Roumeuf, *Shortest paths under rational constraints*, Information Processing Letters, 28 (1988), pp. 245–248.

[SJB97] K. Scott, G. Pabon-Jimenez and D. Bernstein, *Finding alternatives to the best path*, Proc. 76th Annual Meeting of The Transportation Research Board, Washington, D.C. Paper No. 970682, Jan. 1997. Also available as Draft Report *Intelligent Transport Systems Program*, Princeton University, 1997.

[Str94] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity*, Birkhäuser, 1994.

[TR+95a] C. Barrett, K. Birkbigler, L. Smith, V. Loose, R. Beckman, J. Davis, D. Roberts and M. Williams, *An operational description of TRANSIMS*, Technical Report, LA-UR-95-2393, Los Alamos National Laboratory, 1995.

[TR+95b] L. Smith, R. Beckman, K. Baggerly, D. Anson and M. Williams, *Overview of TRANSIMS, the TRansportation ANalysis and SIMulation System* Technical Report, LA-UR-95-1641, Los Alamos National Laboratory, May, 1995.

[Ta81] R. Tarjan, *A unified approach to path problems*, J. ACM, 28-3 (1981), pp. 577–593.

[OR90] A. Orda and R. Rom, *Shortest path and minimium delay algorithms in networks with time dependent edge lengths*, J. ACM, 37-3 (1990), pp. 607–625.

[RR96] G. Ramalingam and T. Reps, *An incremental algorithm for a generalization of the shortest-path problem*, J. Algorithms, 21-2 (1996), pp. 267–305.

[Ya90] M. Yannakakis *Graph theoretic methods in database theory*, invited talk, Proc. 9th ACM SIGACT-SIGMOD-SIGART Symposium on Database Systems (ACM-PODS), Nashville T-N, 1990, pp. 230–242.

[Ya95] M. Yannakakis *Perspectives in database theory*, invited talk, Proc. 36th Annual IEEE Symposium on Foundation of Computer Science (FOCS) 1995, pp. 224-246.