CONF-980119--

LA-UR- 97-2620

Approved for public release; distribution is unlimited.

Title: FORMAL LANGUAGE CONSTRAINED PATH PROBLEMS

Author(s): CHRISTOPHER L. BARRETT
RIKO JACOB
MADHAV MARATHE

Submitted to: ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS (SODA)
SAN FRANCISCO, CA
JANUARY 1998

Los Alamos
NATIONAL LABORATORY

DISCLAIMER

Formal Language Constrained Path Problems

CHRIS BARRETT [1] RIKO JACOB [1] MADHAV MARATHE [1]

July 7, 1997

Abstract

In many path finding problems arising in practice, certain patterns of edge/vertex labels in the labeled graph being traversed are allowed/preferred, while others are disallowed. Motivated by such applications in intermodal transportation planning, we investigate the complexity of finding feasible paths in a labeled network, where the mode choice for each traveler is specified by a formal language. A prototypical problem considered is the following:

> (**Formal Language Constrained Shortest/Simple Paths:**) Given a labeled weighted (directed or undirected) graph G, a source and a terminal s and t, and a formal language L (such as regular, context free, etc) find a shortest (or simple) path from s to t such that the label of the path is in L.

The main contributions of this paper include the following.

1. We show that the problem of finding a shortest path between a source and destination for a traveler whose mode choice is specified as a context free language is solvable efficiently in polynomial time. When the mode choice 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 *simple* paths between a source and a given destination is **NP**-hard, even when restricted to very simple, regular expressions and/or very simple graphs (e.g. interval graphs, meshes, complete graphs, etc.)
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 a destination is solvable in polynomial time and (ii) the extension to finding context free language constrained simple paths is **NP**-complete.

Several extensions of these results are presented in the context of finding shortest paths with additional constraints. These results significantly extend and generalize the results in [MW95]. 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.

Keywords: Transportation Planning, Algorithms, Shortest Paths, Multi-criteria Problems, Formal Languages, Computational Complexity.

---

[1]Email: {barrett,jacob,marathe}@lanl.gov. Los Alamos National Laboratory, P.O. Box 1663, MS P997, Los Alamos, NM 87545. Research supported by the Department of Energy under Contract W-7405-ENG-36.

Formal Language Constrained Path Problems

CHRIS BARRETT [1] RIKO JACOB [1] MADHAV MARATHE [1]

July 8, 1997

Abstract

In many path finding problems arising in practice, certain patterns of edge/vertex labels in the labeled graph being traversed are allowed/preferred, while others are disallowed. Motivated by such applications as intermodal transportation planning, we investigate the complexity of finding feasible paths in a labeled network, where the mode choice for each traveler is specified by a formal language. A prototypical problem considered is the following:

> **(Formal Language Constrained Shortest/Simple Paths:)** Given a labeled weighted (directed or undirected) graph G, a source and a destination s and d, and a formal language L (such as regular, context free, etc) find a shortest (or simple) path from s to d such that the label of the path is in L.

The main contributions of this paper include the following.

1. We show that the problem of finding a shortest path between a source and destination for a traveler whose mode choice is specified as a context free language is solvable efficiently in polynomial time. When the mode choice 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 *simple* paths between a source and a given destination is **NP**-hard, even when restricted to very simple regular expressions and/or very simple graphs (e.g. interval graphs, meshes, complete graphs, etc.)
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 a destination is solvable in polynomial time and (ii) the extension to finding context free language constrained simple paths is **NP**-complete.

Several extensions of these results are presented in the context of finding shortest paths with additional constraints. These results significantly extend the results in [MW95]. 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.

Keywords: Transportation Planning, Algorithms, Shortest Paths, Multi-criteria Problems, Formal Languages, Computational Complexity.

---

[1]Email: {barrett,jacob,marathe}@lanl.gov. Los Alamos National Laboratory, P.O. Box 1663, MS P997, Los Alamos, NM 87545. Research supported by the Department of Energy under Contract W-7405-ENG-36.

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 transport 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 arise 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**[2] project at the Los Alamos 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 expressions.

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 to the reader to the work of [MW95, AMM97, Ha88] for a more thorough discussion on this topic. The following example is essentially from [AMM97].

[2]TRANSIMS is an acronym for the "TRansportation ANalysis and SIMulation System".

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 most case considered in this paper, the label is drawn from a fixed finite set of alphabet Σ. The attribute $w(e)$ denotes the weight of an edge. Again, for this paper, 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 $u = \text{head}(e_1), v = \text{tail}(e_k)$ and $\text{tail}(e_{i-1}) = \text{head}(e_i)$, $1 \leq i \leq k$. 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 $\sum_{1 \leq i \leq k} w(e_i)$ and the label of p is defined as $l(e_1) \cdot l(e_2) \ldots l(e_k)$. In other words the label of a path is obtained by simply concatenating the labels of the edges on the path in order. Let $w(p)$ and $l(p)$ denote the weight and the label of p respectively.

Definition 2.1 Formal Language Constrained Shortest Path:
Given a directed labeled, weighted, graph G, a source destination pair s, 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 2.2 Formal Language Constrained Simple Path:
Given a directed labeled, weighted, graph G, a source destination pair s, d and a formal language (regular, context free, context sensitive, etc.,) L find a 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 NFA-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 NFA-SIP, CFG-SIP and CSG-SIP respectively.

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 in the case of *label constraint* shortest paths. 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:

Here, we investigate the problem of formal language constrained path problems. A number of variants of the problem are considered and both polynomial time efficient algorithms as well as hardness results (**NP**-, **PSPACE**-hardness, etc) are proved. The main results obtained in the paper include :

1. We show that CFG-SHP has a polynomial time algorithm. For NFA-SHP, we give polynomial time algorithms that are substantially more efficient in terms of time and space. The polynomial time solvability holds for the NFA-SHP problem, when the underlying regular expressions are composed of $(\cup, \cdot, *, 2)$[3] operators. We also observe that the extension to regular expressions with $(\cup, \cdot, *, -)$ is **PSPACE**-hard.

[3] operator $\Box^2$ or simple 2 stands for the square operator. R^2 denotes $R \cdot R$.

2. In contrast to the results for shortest paths, we show that the problem finding *simple* paths between a source and a given destination is **NP**-hard, even when restricted to very simple, regular expressions and/or very simple graphs. The problems continue to be **NP**-hard even for fixed regular expressions and for *undirected graphs*. Our intuition is, that this problem is closely related to HAMILTONIAN PATH problem. Using distinct labels, we can essentially choose the set of edges present — this idea can be used to show that NFA-SIP remains **NP**-hard even for simple classes of graphs such as complete graphs, meshes, etc.
3. In contrast to the results in (1) and (2) above, we show that for the class of treewidth bounded graphs, (i) the NFA-SIP is solvable in polynomial time, but (ii) CFG-SIP problem is **NP**-complete. The easiness proof uses a dynamic programming method; although the tables turn out to be quite intricate. The hardness result uses new technique for simulating certain context free grammars by directed graphs. We believe that this technique is of independent interest.
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*.
5. As an application of the theory developed here, we provide a polynomial time algorithm for the BEST k-SIMILAR PATH problem studied in [SJB97]. In [SJB97], the authors present an integer linear program formulation for this problem. They also remark and we quote

   > 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 indirectly implying that BEST k-SIMILAR PATH is **NP**-hard. They then present heuristics based on Lagrangian relaxation method. Here we show that this problem is solvable in time $O(kT_{sp})$ where T_{sp} denotes the time taken to find a shortest path. This substantially improves on the exponential time algorithm of [SJB97].

4 Related Work and Significance

As mentioned earlier, we refer the reader to [TR+95a, AB+97, TR+95b] for a more detailed account of the project. Regular expression constrained shortest path problems were considered by Mendelzon and Wood [MW95]. The authors investigated this problem in the context of finding efficient algorithms for processing database queries (see [CMW87, CMW88, MW95]). 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] gives an incremental algorithm for a version of the grammar problem and as corollaries obtain incremental algorithms for single source shortest path problems with positive edge weights. The results and the models presented here extend/improve the known results in the following ways

1. To our knowledge this is the first attempt to use formal language theory for modeling mode/route choices in transportation science.
2. The results in this paper provide an interesting contrast between the complexity of formal language constrained shortest paths and simple paths.
3. Our results significantly extend the known hardness as well as easiness results in [MW95], on finding regular expression constrained simple paths. In [MW95] the authors state

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

The hardness and the polynomial time algorithms presented here are a step towards finding such classes of graphs for which polynomial time query evaluation is possible.

4. 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.
5. The results in (1) (2) and (3) provide fairly tight bounds on the polynomial time solvability of the problems studied, when the underlying formal language varies. Note that while CFG-SHP is plausibly easier to solve that CFG-SIP, CSG-SHP is "harder" than CSG-SIP— depicting an interesting contrast between the complexities of these problems.
6. Finally, our theory also allows us to solve other related route planning problems that arise in practice. As an example consider the trip chaining problem. In this problem, we need to find a shortest route that visits a sequence of destination types. For example a typical trip might look like — (home-daycare-shop-postoffice-cafe-home). In the above example, there are many options for shops, cafe and postoffice and our goal of course is to find the best combination of them. By appropriately constructing regular expressions and extending our ideas to node labels, we can solve this problem more efficiently than the naive way of keeping all possible shortest path estimates between the sub-destination types.

		RL	CFL	CSL
shortest path	general graph	$O(\|R\|\|G\| \log(\|R\|\|G\|))$	**FP**	undecidable
	directed chain	linear	**FP**	undecidable
simple path	planar/grid graph	**NP**-c.	**NP**-c.	**PSPACE**-c.
	treewidth bounded	**FP**	**NP**-c.	**PSPACE**-c.
	directed chain	linear	cubic	**PSPACE**-c.

Figure 1: Summary of results on formal language constrained simple/shortest paths. RL, CFL, CSL denote regular, context free and context sensitive languages respectively. For regular languages, the polynomial time results hold for regular expressions with the operators $(\cup, \cdot, *, 2)$. **FP** states that the sought path can be computed in deterministic polynomial time, even if the language specification is part of the input. On the other hand most of the hardness results are valid for fixed languages. A directed chain is a graph that forms exactly one directed simple path.

The rest of the paper contains selected proof sketches. Additional results and proofs can be found in the appendix. Definitions and basic concepts in formal language theory and computational complexity theory can be found in [HU79, GJ79]

5 Results for Shortest Path Problems

In this section, we present polynomial time algorithms for the problems NFA-SHP and CFG-SHP.

5.1 Algorithm for NFA-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[4] 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.

[4]NFA stands for nondeterministic finite automata

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

Definition 5.1 *Given a label directed graph G, a source and a destination s and d define an 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. *$j \in \delta(i, a)$ if there is an edge (i, j) with label a.*

Definition 5.2 *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 automaton 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.

ALGORITHM RE-CONSTRAINED-SHORT-PATHS:

- *Input:* A regular expression R, a directed labeled weighted graph G, a source and destination pair s and d.
- 1. Construct an NFA $M(R) = (S, \Sigma, \delta, s_0, F)$ from R.
  2. Construct an NFA $M(G)$ of G.
  3. Construct $M(G) \times M(R)$. The length of the edges in the product graph are to the according edges in G.
  4. Starting from state (s_0, s), find a shortest path to all of the vertices (f, d), where $f \in F$. Denote these paths by p_i, $1 \leq i \leq w$. Also denote the cost of p_i by $w(p_i)$
  5. $C^* := \min_{p_i} w(p_i)$; 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 5.3 ALGORITHM RE-CONSTRAINED-SHORT-PATHS *computes exact solution for the problem* NFA-SHP *with non-negative edge weights in time $O(|R||G| \log(|R||G|))$.*

Proof: The correctness of the algorithm follows by observing the following:

Consider a shortest path q^* in G of cost $w(q^*)$, that satisfies the regular expression R. Then there is 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 in $M(G) \times M(R)$ of cost $w(\tau)$, that begins on a starting state and ends on a final state, there is 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 $O(|R||G|)$. Using efficient algorithms to solve shortest path problems in Step 4 the problem can be solved in time $O(|R||G| \log(|R||G|))$. □

5.2 Extensions

We consider two possible extensions for the problem NFA-SHP– (i) regular expressions with the operators $\cup, \cdot, *, 2$ and regular expressions consisting of $\cup, \cdot, *, -$, i.e. union, dot, star and complement. We refer the reader to [HU79] for the corresponding definitions. Let us denote these problems by NFA-SQUARE-SHP and NFA-COMPLEMENT-SHP respectively. First consider the problem NFA-SQUARE-SHP. A simple extension of the algorithm given above for NFA-SHP does not work because of the following reason. The NFA which needed to represent such a regular expression can in general be exponentially larger than the regular expression – implying that the algorithm could take exponential time to execute in the worst case. We show that a slightly different way of viewing our technique allows us to obtain polynomial time algorithm for NFA-SQUARE-SHP.

We can solve these problems by applying a dynamic programming approach, that computes tables of all pairs constrained shortest paths in a bottom up way along the regular expression.

Let R_1 and R_2 be regular expressions. Assume that for each regular expression we have a table telling us for all pairs of network nodes, the length of the shortest path according to the regular expression. We will show how to compute the new table for the compound expression.

Case 1: Let the operator be $\cup$. The language accepted is $L(R_1) \cup L(R_2)$. In this case, the new table consists of the component wise minimum of the two original tables.

Case 2: Let the operator be $\cdot$. Then for every (s, d) pair the minimum over all intermediate vertices i of the sums of subpaths is computed.

Case 3: Let the operator be $\square^2$. The same as **Case 2**, only that the tables are identical.

Case 4: Let the operator be $*$. We know that $R^* = \varepsilon \cup R \cup R^2 \cup \cdots$. Since we are looking for shortest paths in the graph (without negative cycles), R^k for $k > n$ is never used. As a result, we can use the ideas in **Case 1-3** to solve this case.

Next, consider regular expressions with $(\cup, \cdot, *, -)$ as operators. By a simple reduction from the problem REGULAR EXPRESSION INEQUIVALENCE (See [GJ79] problem AL9) we can show that

Proposition 5.4 NFA-SHP *is* **PSPACE**-*hard for regular expressions over* $(\cup, \cdot, *, -)$.

Given an instance R of REGULAR EXPRESSION INEQUIVALENCE problem, we create a graph G with one node v. There is a self loop from v to v for each symbol in the alphabet. There exists a path from v to v to with a label in $L(R)$ if and only if $\Sigma^* - L(R) \neq \emptyset$, i.e. $L(R) \neq \Sigma^*$. □

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. Such an extension is significant for the following reasons (i) The result stands in contrast to the hardness of simple path problems even for regular expression, (ii) it allows us to work with more powerful queries in database systems and allow natural mode choices to be included, (iii) it strengthens the bounds on what problems are feasible. For example, a traveler might prefer a freeway over a arterial. Unfortunately, using regular expressions, we cannot specify mode choices that enforce the travel distance over a freeway to be at least say 5 times the travel distance on arterials. This can be specified using expressions such as $a^n f^{5n+5m} f^* a^m$.

The algorithm for solving context free grammar constrained shortest paths is a dynamic programming based approach. Hence we will first investigate the structure of an optimal shortest path from s to d in G that obeys CFG R constraints. Assume that R is in Chomsky normal form (see [HU79] for definition). Consider any such shortest path p such that $l(p) = a_1, a_2, \ldots a_m$. One important property of 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 can be decomposed into two parts l_1 and l_2 such that $l(p) = l_1 l_2$, $S \to AB$, $A \to^* l_1$ and $B \to^* l_2$.

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:

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

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

Observation 5.5 *These equations uniquely determine the function D, and they immediately imply a polynomial time dynamic programming approach.*

Proof: See appendix.

6 Constrained Simple Paths

Next, we investigate the complexity of finding formal language constrained simple paths. The question of CFG-shortest-simple paths on graphs with bounded treewidth turns out to be **NP**-hard.

Before formally stating the proof, we give the overall idea. We present a reduction from 3SAT (See e.g. [GJ79] for definition). The basic idea is to have a path, that in the first half uniquely chooses an assignment, and then in the second part 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.

Formally, $L = L_1 \cap L_2$, where $L_1 = \{ w_0\#w_1\$w_1^R\#w_2\$w_2^R t \dots w_k\$w_k^R\#w_{k+1} \mid w_i \in \Sigma^* \}$ 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 $\forall i$, $w_i^R = v_{i+1}$, $v_{i+1}^R = w_{i+1}$, establishing that $\forall i$, $v_i = v_{i+1}$ and thus $L = L_1 \cap L_2$.

Now, imagine w representing an assignment to the variables of a 3CNF 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 straight forward 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 the act of 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 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 is easy and is omitted.

Theorem 6.1 *The* CFG-SIP *problem is* **NP**-*hard, even for Graphs of constant treewidth and a fixed Context Free Language.*

Proof: Reduction from 3SAT. Let $F(X, C)$ be a 3CNF formula, where $X = \{x_1, \dots, x_n\}$ denote the set of variables and $C = \{c_1, \dots, c_m\}$ denote 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) goes 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, and is depicted in Figure 2.

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 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 3. Before we describe the remaining edges, we discuss the properties of a tower.

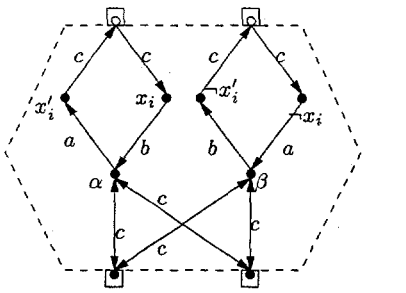

Figure 2: the gadget H_i used to implement the tower, forcing the assignment to be spread consistently over the graph.

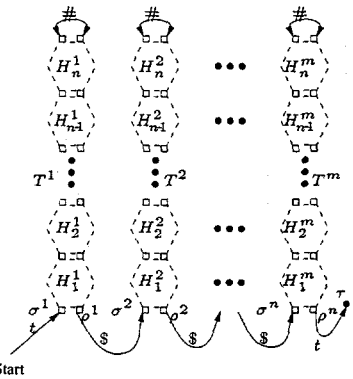

Figure 3: Assembling the gadgets, building the first part of the graph.

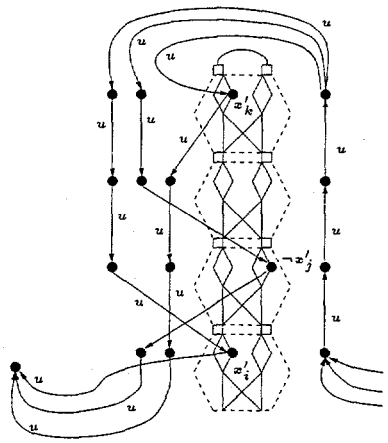

Figure 4: One tower of the second part of the graph, according to the clause $(\neg x_i \vee x_j \vee \neg x_k)$.

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

1. The path consists of two simple sub-paths p and q with an edge labeled $\#$ in between. p starts at the bottom of the tower and is labeled according to the regular expression $l(p) = (c(a \cup b)c)^*$. 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. Further q is *uniquely* determined given p. Observe that the label of this sub-path is uniquely determined by the label of the upward path, it is its reverse.
3. Note that for each variable x_i, the simple path visits exactly one of the nodes x_i and $\neg x_i$, If the path reflects the assignment that 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 that this is indeed a satisfying assignment for F.

Proposition 6.2 *Any simple path starting at the start node s, and reaching the intermediate node τ, with the constraint that the labeling belongs to $t((c(a \cup b)c)^*\#(c(a \cup b)c)^*\$)^*t$ generates the language*

$$L_2 = \{ tw_1\#w_1^R\$w_2\#w_2^R\$ \ldots w_n\#w_n^Rt \mid w_i \in (c(a \cup b)c)^* \}.$$

We now choose the CFL to be
$L_1 = \{ tw_1\#w_2\$w_2^R\#w_3\$w_3^R\# \ldots w_{k-1}\$w_{k-1}^R\#w_kt \mid k \in \mathbb{N}, w_i \in (ccc(a \cup b))^* \}$.
The following important lemma follows from Proposition 6.2 and definition of L_1.

Lemma 6.3

1. *Proposition 6.2 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 assignment 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 4. Each of this D_i basically consists of four simple chains. The first one goes up the tower. There it is split into three downward directed subpaths, for each literal one path. Three connecting edges to subgraph D_{i+1} join these paths again. On this way down every path uses the node that corresponds to its literal. The vertex τ is used instead of D_0, instead of D_{m+1} the destination vertex d is used. This completes the description of the graph $G(V, E_1 \cup E_2)$.

The instance of CFG-SIP consists of the graph $G(V, E_1 \cup E_2)$ and the CFL is $L_1 \cdot u^*$. 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. The complete proof of correctness is omitted due to lack of space. It is easy to see that G has a bounded treewidth (can be shown to be 16). □

Another extension considered is to investigate both class of problems (simple and shortest paths) for undirected networks. We extend the result of [MW95] to hold for *undirected* graphs. Such graphs adequately model problems in hypertext retrieval and VLSI designs. The extension was not considered in [MW95], and indications were given that implied that the answer is not clear. The following two theorems whose proofs can be found in the Appendix, summarize the new hardness results for the NFA-SIP problems.

Theorem 6.4 *The problem of computing Regular Expression Constrained Simple Paths is* **NP***-complete even for undirected graphs, and for the fixed regular expression $(abab)^*$.*

Theorem 6.5 *Let C be a graph class (such as planar, grid, etc) such that the* HAMILTONIAN PATH *problem is* **NP***-hard when restricted to C. The problem of finding regular expression constrained simple paths is* **NP***-hard when restricted to C.*

The result immediately implies the hardness of finding simple paths in grid graphs, bipartite graphs, etc. Note that in this case the regular expression depended on the input graph G.

Given the hardness results for solving NFA-SIP problems even for fixed "simple" regular expressions, the natural question to investigate is — are there natural restrictions that can be placed on the class of regular grammars or the input graphs to yield polynomial time solutions.

[MW95], initiated this line of investigation and defined the notion of *restricted regular expressions*. Given a regular expression R, let R' be the regular expression obtained by replacing some occurrence of a symbol $a \in \Sigma$ in R by $(a \cup \varepsilon)$. Then R is said to be *restricted* if and only if $R \equiv R'$ for any R' obtained as outlined above. Mendelzon and Wood [MW95] show that the NFA-SIP problem is solvable in polynomial time for restricted regular expression. Motivated by the remark in [MW95] discussed in Section 4 we investigate the complexity of the NFA-SIP problem for restricted classes of graphs. Our results show that in general the problem continues to remain **NP**-hard even for the simplest classes of graphs.

Theorem 6.6 *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* either *(i) a k-clique,* or *(ii) a $k \times k$-mesh, then the* NFA-SIP *problem is* **NP**-*hard for $\mathcal{C}$.*

Thus the NFA-SP *problem is* **NP**-*hard even for (i) interval graphs, (ii) Chordal graphs, (iii) complete meshes, (iv) Complete hypercubes, (v) permutation graphs.*

Proof: See appendix.

In contrast to these **NP**-hardness results, we show that if the graphs have bounded treewidth then the problem is polynomial time solvable. It should be noted that this includes (i) interval graphs and Chordal graphs with *bounded clique size*, (iii) complete meshes, with fixed length or width, and other well known classes. The hardness result above also imply that these results are in a sense the best possible. The proof of the following theorem can be found in the appendix:

Theorem 6.7 *The* NFA-SIP *problem is solvable in polynomial time on treewidth bounded graphs*

An easy extension of this construction shows, that the pathwidth can also be bounded by a constant.

7 Extensions and Applications

7.1 Node Labeling

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. Details appear in the appendix.

7.2 Finding Alternatives to Shortest Paths

There has been a considerable interest in finding 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. Our approach for solving the problem is based on using our algorithm for regular expression based shortest paths. The approach uses the fact that for fixed k the words, that have less or equal to k symbols a in them, is regular. More details are given in the appendix.

7.3 Other Formal Languages

As expected attempts to extend the techniques to more general grammars such as the context sensitive grammars fails to yield polynomial time results. Intuitively, the hardness of the problems is due to the fact that *recognition* or *emptiness problems* for such grammars are hard. Consider for example the problem for context sensitive grammars. It is easy to show that the problem (CSG-SHP) is undecidable. The proof is straightforward and thus we give only the essential features. Given a fixed CSG R, it is well known that the question whether $L(R) \neq \emptyset$ is undecidable. We reduce this problem to the problem of finding paths. Given R we simply create a graph G with one node and self loop for each symbol in the alphabet. There exists a path from this node to itself with the labeling in the language if and only if $L(R) \neq \emptyset$.

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.

Acknowledgments: We thank Harry B. Hunt III, Sven O. Krumke, S.S. Ravi, Ravi Sundaram, R. Ravi, Venkatesh Radhakrishnan, Daniel Rosenkrantz and Hans-Christoph Wirth for several useful discussions and suggested improvements in the course of writing this paper. We also thank the members of the research group at **TRANSIMS**, in particular, Kai Nagel, Terence Kelly, Robert White and Brian Bush for useful conversations. We are also thankful to S.S. Ravi and Terence Kelly for drawing our attention to the work of [MW95] and [SJB97].

References

[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)*, vol. 40:5, pp. 1134-1164 (1993).

[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, Feb 97.

[AL+91] S. Arnborg, J. Lagergren and D. Seese, "Easy Problems for Tree-Decomposable Graphs," *Journal of Algorithms*, vol. 12, pp. 308-340 (1991).

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

[Bo92] H. Bodlaender, "A tourist guide through treewidth," *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," *Journal of Algorithms*, vol. 8, pp. 216-235 (1987).

[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 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.

[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, January 1997. Also available as Draft Report *Intelligent Transport Systems Program*, Princeton University, 1997.

[MW95] A. Mendelzon and P. Wood, "Finding Regular Simple Paths in Graph Databases," *SIAM J. Computing*, vol. 24, No. 6, 1995, pp. 1235-1258.

[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*, 1990, pp. 22-31.

[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.

[OR90] A. Orda and R. Rom, "Shortest Path and Minimium Delay Algorithms in Networks with Time Dependent Edge Lengths," *J. ACM* Vol. 37, No. 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):267-305, September 1996.

[Ku77] D.E. Knuth, "A Generalization of Dijkstra's Algorithm," *Information Proc. Lett.*, 6(1), pp. 1-5, 1977.

8 Appendix

8.1 Basic Definitions

Definition 8.1 *Let Σ be a finite alphabet disjoint from $\{\varepsilon, \phi, (,)\}$. 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" is an atomic regular expression.*
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.*

As mentioned earlier, in a regular expression, ε denotes the empty string, ϕ denotes the empty set, $\cup$ denotes union, $\cdot$ denotes concatenation and $*$ denotes reflexive transitive closure under concatenation. We will also use R^+ to denote the positive closure of R under $\cdot$.

Definition 8.2 *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. $L(\varepsilon) = \{\varepsilon\};\ L(\phi) = \{\phi\};\ \forall a \in \Sigma\colon L(a) = \{a\}$
2. $L(R_1 \cup R_2) = L(R_1) \cup L(R_2) = \{\, w \mid w \in L(R_1) \text{ or } w \in L(R_2) \,\}$
3. $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2) = \{\, w_1 w_2 \mid w_1 \in L(R_1) \text{ and } w_2 \in L(R_2) \,\}$
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 8.3 *A* nondeterministic finite automaton *(NFA) M is a 5 tuple $(S, \Sigma, \delta, s_0, F)$, 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 as $|S||\Sigma|$.

Definition 8.4 *A $M = (S, \Sigma, \delta, s_0, F)$, be a NFA. The* language accepted *by M denoted $L(M)$, is the set*

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

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

Definition 8.5 *A* context free grammar *(CFG) G is a 4 tuple (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 8.6 Treewidth bounded graphs: *The class of k-trees are recursively defined as follows.*

A clique of size $k+1$ is a k-tree.

A k-tree with $n+1$ vertices can be obtained from a k-tree with n vertices by adding a new vertex and edges from the new vertex to a set of k completely connected vertices.

A **partial k-tree** *is a subgraph of a k-tree. The minimum value of k for which a graph is a subgraph of a k-tree is called the treewidth of the graph*
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 a graph is the minimum treewidth of a tree decomposition.

A class of treewidth-bounded graphs can be specified using a finite number of primitive graphs and a finite collection of binary composition rules. We use this characterization for proving our results. A class of treewidth-bounded graphs Γ is inductively defined as follows [BL+87].

1. The number of primitive graphs in Γ is finite.
2. Each graph in Γ has an ordered set of special nodes called **terminals**. The number of terminals in each graph is bounded by a constant, say k.
3. There is a finite collection of binary composition rules that operate only at terminals, either by identifying two terminals or adding an edge between terminals. A composition rule also determines the terminals of the resulting graph, which must be a subset of the terminals of the two graphs being composed.

8.2 CFG-SHP

Proof of Observation 5.5
Consider the case, that D and D' satisfy the above recurrence but are different. Then there must be a smallest witness of this fact $a = D(i, j, A)$ and $b = D'(i, j, A)$, and $a \neq b$. Let us assume $a > b$ and consider the smallest such b.

By the definition we know that A is a nonterminal, b is finite and satisfies (1). This and the fact that all length are positive, imply that there must exist two witnesses that establish this minimum as their sum. These have values less then b. On these D and D' are identical, otherwise a and b would be chosen different. Because of that, a can not satisfy (1).

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. The proof then implies, that at least one entry every round, namely that with the smallest value that changed, is finally labeled.

This Algorithm would take time at most one round for every entry, by this the square of the number of entries in the table. This is polynomial, namely $O(|V|^4|G|^2|R|)$ with $|G|$ denoting the number of Nonterminals and R the number of rules. (Recall that G is in Chomsky normal form.)

Another implementation would be to fill the table with a Dijkstra type algorithm. There a heap is used to hold the current estimates on entries, and the smallest one is finally put in the table generating a lot of new

estimates. We would have for every entry one extract-min operation and up to $2|V|$ update operations. Using Fibonacci-Heaps (see [CLR] for an analysis), this sums up to $O(|V|^2|G| \cdot (\log(|V|^2|G|) + 2|V||R|))$, that is $O(|V|^3|G||R|)$.

Proof of theorem 6.7:
We use the fact, that we can move separators over the graph that stay bounded because of the treewidth bound. We will use the notion of a nice tree decomposition.

Definition 8.7 *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* $X(i) = X(j) = X(k)$*;*
- *if* i *has one child, 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].

We will describe an algorithm, that computes tables of (partial) shortest simple paths bottom up in the tree.

As we are talking about simple paths, we have to take care of used nodes. Because of that we must carry with us all possibilities to enter and leave a visited region. This is in general far too much information, but as the treewidth is bounded, we can do it.

Dividing paths in the network into subpaths, we have to do this also with paths in the automaton. As these paths don't have to be simple things are easier. If we remember the starting and ending vertex of such a path, we know everything we need to know.

We will maintain tables for sets of active nodes. The indices are made up of the following information:

- $k \geq 0$ denoting the number of subpaths,
- k ordered pairs of active nodes and k ordered pairs of states in the NFA, denoting simple subpaths in the already visited graph, (identical nodes are permitted for the case of paths of length zero)
- the subset of current nodes, that are used, excluding endpoints of the subpaths.
- possibly an active node, and a state for a simple path coming from s to this node labeled such that the automaton is in this state,
- possibly an active node, and a state for a simple path leading to t such that the automaton can reach a final state from this state.

Note that the entry for $k = 0$ and none of the last two items, represents a simple path from s to t in the already visited graph.

This index set is polynomial in the size of the NFA.

We maintain for each index the summed up length of all the partial paths. It is important, that they could get concatenated to a simple path, i.e. that they do not use any node in the visited graph twice.

We will now use the nice tree decomposition to provide sets of current nodes, that only change a little bit.

For the leaf node the table is easy to compute. It has 0-entries at the following indices:
for all states i: (1 path, this node to this node with (i, i), this node used, -, -);
if it is the start node: (0 paths, - , this node used, ε, -) and a similar, if it is the destination node.

Join
- keep the component wise min of the tables,
- make all possible concatenations. That is either if the sets of used nodes are disjoint and the start-path and the destination path is in at most one of the indices, or if the partial paths can get concatenated at one or more vertices and the sets of used nodes are otherwise disjoint.

For this, we essentially have to try out all pairs of X_1-X_2 indices, see if they fit together somehow, and if create the appropriate entry of the table with the sum of weights. If this entry already exists, we only update it, if the new entry would be smaller than the old one.

Forget We eliminate the entries in the table, where this node is used as endpoint in the index. We delete this node in all sets of used nodes in all indices.

Introduce we add the edges one by one.

1. Set up new node. Extend table by all indices extended by this node used as an ε path, or if it is start or destination vertex specially.
2. Include all edges one by one. Try to close table entries with this edge, by this creating new entries. If both of the endpoints of the edge are used as appropriate endpoints of subpaths, and the labeling matches the transition in the automaton, then there exists a new path not being open at these points. We create or update the according entry of the table.

The proof of correctness goes as follows: First, note that no entry of the table is ever to small, i.e. that whenever there is a table entry, there also is an according path of this type and the given length.

On the other hand, we take (one of) the shortest paths and follow the tables throughout the run of the algorithm. We find, that whenever a part of the paths is already visited, an appropriate entry in the table exists. □

8.3 Hardness for simple graph classes

Proof of Theorem 6.4
The reduction is a simple modification from the problem of finding simple paths in *directed graphs*, for the regular expression $(00)^*$. We simply replace each directed edge (u, v) in G by two undirected edges $(ut_{uv}), (t_{u,v}, v)$ to obtain a new graph G_1. The label on edge (ut_{uv}) is a, and the label on edge $(t_{u,v}, v)$ is b. Now we claim that there is a simple path from x to y in G with label of the form $(00)^*$ if and only if there is a simple path from x to y with label $(abab)^*$. The if part is easy and so we only briefly discuss the converse. Note first that G_1 is bipartite and hence any simple path going from x to y will look like $x - t_{i_1} - x_{i_1} - t_{i_2} - x_{i_2} \ldots t_{i_k} - y$, where the vertices $x'_{i_j}s$ are the vertices in G and t_{i_j} are the new vertices added. Also note that each t_{i_j} has degree exactly 2. This immediately implies that the path can be transformed to the appropriate in G. □

As we mentioned earlier, the hardness of the simple path problem stems from the Hamiltonian path type nature of the problem. We formalize this intuition in the following.

Proof sketch of theorem 6.5
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 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 is an only if there is a simple path in G_1 that satisfies a^n. □

Proof sketch of theorem 6.6 The basic idea of the reduction is the following. We assume that given a graph G, two vertices s and t the problem of finding a Hamiltonian path starting at s and ending at t is **NP**-hard (see [GJ79], Problem GT39). Given G, s and t, we create a complete graph G_1 as follows: All the edges

in G_1 that were in G are given labels a, the remaining edges are given label b. Now we claim that G_1 has a simple path from s to t with label a^n if and only if G has a Hamiltonian path starting at s and ending at t.

Since a complete graph is also an (i) interval graph , (ii) Chordal graph (iii) perfect graph, etc, we get the required result.

A similar embedding can be done by starting with G being a grid graph and constructing a complete grid G_1 out of it.

These reductions can be done in polynomial time. □

8.4 Node Labeling

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 edge symbol and (ii) The word without the edge symbols is in the original Language.

The regular and context free languages are closed under this operation. This shows, that easiness results for edge label constraints transfer into easiness results for node label constraints.

If we have a 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 transfers edge label constraints hardness results to node label constraints.

8.5 k-Similar Path

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 is shown in Figure 5.
3. Find a shortest path in G with the constraint that its label is in $L(M)$.

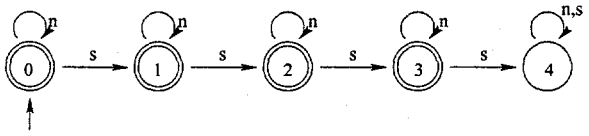

Figure 5: Automaton used to count shared links, accepting if less than 4

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.

The question which naturally comes up is – what happens if the first path is not given? In such a case, it is easy to see that for general case (i.e. setting $k = 0$) the problem reduces to finding two disjoint paths of small total length – a well known hard problem. By simple padding arguments, the hardness result can be extended to bigger values of k.