HAL open science

Constructing Strings Avoiding Forbidden Substrings

Giulia Bernardini, Alberto Marchetti-Spaccamela, Solon P. Pissis, Leen Stougie, Michelle Sweering

To cite this version:

Giulia Bernardini, Alberto Marchetti-Spaccamela, Solon P. Pissis, Leen Stougie, Michelle Sweering. Constructing Strings Avoiding Forbidden Substrings. CPM 2021 - 32nd Annual Symposium on Combinatorial Pattern Matching, Jul 2021, Wroclaw, Poland. pp.1-18. hal-03395386

HAL Id: hal-03395386

https://inria.hal.science/hal-03395386

Submitted on 22 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Constructing Strings Avoiding Forbidden Substrings

Giulia Bernardini
CWI, Amsterdam, The Netherlands

Alberto Marchetti-Spaccamela
Dept. of Computer, Automatic and Management Engineering, Sapienza University of Rome, Italy
ERABLE Team, Lyon, France

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands
ERABLE Team, Lyon, France

Leen Stougie
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands
ERABLE Team, Lyon, France

Michelle Sweering[1]
CWI, Amsterdam, The Netherlands

Abstract

We consider the problem of constructing strings over an alphabet Σ that start with a given prefix u, end with a given suffix v, and avoid occurrences of a given set of *forbidden substrings*. In the decision version of the problem, given a set S_k of forbidden substrings, each of length k, over Σ, we are asked to decide whether there exists a string x over Σ such that u is a prefix of x, v is a suffix of x, and no $s \in S_k$ occurs in x. Our first result is an $\mathcal{O}(|u| + |v| + k|S_k|)$-time algorithm to decide this problem. In the more general optimization version of the problem, given a set S of forbidden arbitrary-length substrings over Σ, we are asked to construct a shortest string x over Σ such that u is a prefix of x, v is a suffix of x, and no $s \in S$ occurs in x. Our second result is an $\mathcal{O}(|u| + |v| + ||S|| \cdot |\Sigma|)$-time algorithm to solve this problem, where $||S||$ denotes the total length of the elements of S.

Interestingly, our results can be directly applied to solve the reachability and shortest path problems in complete de Bruijn graphs in the presence of forbidden edges or of forbidden paths.

Our algorithms are motivated by data privacy, and in particular, by the data sanitization process. In the context of strings, sanitization consists in hiding forbidden substrings from a given string by introducing the least amount of spurious information. We consider the following problem. Given a string w of length n over Σ, an integer k, and a set S_k of forbidden substrings, each of length k, over Σ, construct a shortest string y over Σ such that no $s \in S_k$ occurs in y and the sequence of all other length-k fragments occurring in w is a subsequence of the sequence of the length-k fragments occurring in y. Our third result is an $\mathcal{O}(nk|S_k| \cdot |\Sigma|)$-time algorithm to solve this problem.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, forbidden strings, de Bruijn graphs, data sanitization

Digital Object Identifier 10.4230/LIPIcs.CPM.2021.9

Funding *Giulia Bernardini*: Netherlands Organisation for Scientific Research (NWO) under project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)"
Leen Stougie: Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003
Michelle Sweering: Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003

[1] Corresponding author

© Giulia Bernardini, Alberto Marchetti-Spaccamela, Solon P. Pissis, Leen Stougie, and Michelle Sweering;
licensed under Creative Commons License CC-BY 4.0
32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021).
Editors: Paweł Gawrychowski and Tatiana Starikovskaya; Article No. 9; pp. 9:1–9:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We wish to thank Gabriele Fici (Università di Palermo) for bringing to our attention the work of Crochemore, Mignosi and Restivo [21].

1 Introduction

We start with some basic definitions and notation from [20]. An *alphabet* Σ is a finite nonempty set of elements called *letters*. We assume throughout an integer alphabet $\Sigma = [1, |\Sigma|]$. A *string* $x = x[1] \dots x[n]$ is a sequence of *length* $|x| = n$ of letters from Σ. The *empty* string, denoted by ε, is the string of length 0. The fragment $x[i \mathinner{.\,.} j]$ is an *occurrence* of the underlying *substring* $s = x[i] \dots x[j]$; s is a *proper* substring of x if $x \neq s$. We also say that s occurs at *position* i in x. A *prefix* of x is a fragment of x of the form $x[1 \mathinner{.\,.} j]$ and a *suffix* of x is a fragment of x of the form $x[i \mathinner{.\,.} n]$. An *infix* of x is a fragment of x that is neither a prefix nor a suffix. The set of all strings over Σ (including ε) is denoted by Σ^*. The set of all length-k strings over Σ is denoted by Σ^k.

We consider the following basic problem on strings.

> String Existence Avoiding Forbidden Length-k Substrings (SEFS)
> **Input:** An integer $k > 0$, two strings $u, v \in \Sigma^*$, and a set $S_k \subset \Sigma^k$.
> **Output:** YES if there exists a string $x \in \Sigma^*$ such that u is a prefix of x, v is a suffix of x, and no $s \in S_k$ occurs in x; or NO otherwise.

In what follows we refer to set S_k as the set of *forbidden substrings*.

▶ **Example 1.** Consider $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $k = 4$, $S_k = \{\mathtt{bbbb}, \mathtt{aaba}, \mathtt{abba}\}$, $u = \mathtt{aab}$, and $v = \mathtt{aba}$. SEFS has a positive answer, as there exists, for instance, string $x = \mathtt{aabbbaba}$ with u as a prefix, v as a suffix, and with no occurrence of any $s \in S_k$.

In Section 3, we show the following result.

▶ **Theorem 2.** *Given an integer $k > 1$, two strings $u, v \in \Sigma^*$, and a set $S_k \subset \Sigma^k$, SEFS can be solved in $\mathcal{O}(|u| + |v| + k|S_k|)$ time.*

We also consider the following more general optimization version of the SEFS problem.

> Shortest String Avoiding Forbidden Substrings (SSFS)
> **Input:** Two strings $u, v \in \Sigma^*$ and a set $S \subset \Sigma^*$.
> **Output:** A shortest string $x \in \Sigma^*$ such that u is a prefix of x, v is a suffix of x, and no $s \in S$ occurs in x; or FAIL if no such x exists.

Note that in SSFS the set S of forbidden substrings contains strings of arbitrary lengths. In Section 4, we show the following result.

▶ **Theorem 3.** *Given two strings $u, v \in \Sigma^*$, and a set $S \subset \Sigma^*$, SSFS can be solved in $\mathcal{O}(|u| + |v| + ||S|| \cdot |\Sigma|)$ time, where $||S|| = \sum_{s \in S} |s|$, using $\mathcal{O}(|u| + ||S|| \cdot |\Sigma|)$ space.*

Related Work. Crochemore, Mignosi and Restivo [21] showed how to construct a complete deterministic finite automaton (DFA) accepting strings over Σ which do not contain any forbidden substring from a finite anti-factorial language S (see also [8]). This DFA has $\mathcal{O}(||S||)$ states, $\Theta(||S|| \cdot |\Sigma|)$ edges in the worst case, and it can be constructed in $\Theta(||S|| \cdot |\Sigma|)$ time. Thus using this DFA for deciding SEFS would entail a time complexity of $\Omega(|u|+|v|+k|S_k|\cdot|\Sigma|)$ in the worst case. We show a fundamentally different *non-constructive* approach to decide SEFS in $\mathcal{O}(|u| + |v| + k|S_k|)$ time, which is based on combinatorial properties of complete

de Bruijn graphs. For solving the optimization version SSFS, we make use of the DFA complemented with an efficient way to compute the *appropriate* source and sink nodes. This is because u (as a prefix) and v (as a suffix) must occur in string x (possibly overlapping).

Following the definition in [15], a language $L \subseteq \Sigma^*$ is *strictly locally testable* if there exist an integer k and finite sets $F, U, V \subseteq \bigcup_{i=1}^{k-1} \Sigma^i$ and $W \subseteq \Sigma^k$ such that $L = ((U\Sigma^* \cap \Sigma^* V) \setminus \Sigma^* W \Sigma^*) \cup F$. Therefore SEFS can be reduced to determining whether or not some specific strictly locally testable language is nonempty, while SSFS can be reduced to finding a minimum-length string in its corresponding language.

Our Motivation. We are motivated by applications in data privacy, and in particular, in data sanitization. *Data sanitization*, also known as *knowledge hiding*, is a privacy-preserving data mining process, which aims at preventing the mining of confidential knowledge from published datasets. Data sanitization has been an active area of research for the past 25 years [18, 38, 42, 29, 43, 30, 1, 2, 28, 31, 37, 13]. Informally, it is the process of disguising (hiding) confidential information in a given dataset. This process typically incurs some data utility loss that should be minimized. Naturally, privacy constraints and utility objective functions lead to the formulation of combinatorial optimization problems. From a fundamental perspective, it is thus relevant to be able to establish some formal guarantees.

In the context of strings, data sanitization consists in hiding forbidden substrings from a given string by introducing the least amount of spurious information [9, 10, 11, 12]. In previous works [9, 10, 11], we considered various combinatorial optimization problems for string sanitization, all of which receive as input a string w of length n over Σ, an integer k, and a set S_k of forbidden substrings, and conceal the occurrences of forbidden substrings in w through the use of a special letter $\# \notin \Sigma$. In particular, the TFS problem [9] asks to construct a shortest string x such that no string in S_k occurs in x and the order of occurrence of all other length-k substrings over Σ (and thus their frequency) is the same in w and in x. We developed an algorithm that solves the TFS problem in the optimal $\mathcal{O}(n + |x|)$ time [9], assuming that the list of all occurrences of forbidden substrings in w are given at input.

▶ **Example 4.** Let $w = \underline{\texttt{abbbb}}\texttt{a}\underline{\texttt{aaba}}\texttt{a}$, $\Sigma = \{\texttt{a}, \texttt{b}\}$, $k = 4$, and $S_k = \{\texttt{bbbb}, \texttt{aaba}, \texttt{abba}\}$. All occurrences of forbidden substrings are underlined. The solution to the TFS problem is string $x = \texttt{abbbaaab\#abaa}$, where $\# \notin \Sigma$: it is the shortest string in which the occurrences of the strings in S_k are concealed and the order of all other length-k substrings over Σ is preserved.

However, as already noted in [9], the occurrences of # in x may reveal the *locations* of the forbidden substrings in w and should therefore be ultimately replaced by letters or strings over Σ in several applications of interest. The problem of replacing the occurrences of # in x with *single letters* of Σ without reintroducing any forbidden substrings and with different optimization criteria has been considered in [9, 10, 12]. Replacing #'s with single letters, though, may be too restrictive, and even "easy" instances may admit no feasible solution.

▶ **Example 5.** Consider the instance from Example 4. The occurrence of # in $x = \texttt{abbbaaab\#abaa}$ reveals the location in w of the forbidden substring aaba. Note that deleting # would reinstate aaba; and # cannot be replaced by a single letter from Σ, as both a and b create occurrences of the forbidden substrings aaba and abba, respectively.

We thus consider a more general string sanitization problem in which we allow # replacements with strings of *any* length over Σ, so as to widen the set of instances that admit a feasible solution. Given a string w over an alphabet Σ, an integer $k > 1$, and a set $S_k \subset \Sigma^k$, we denote by $\mathcal{S}(w, S_k)$ the sequence over $\Sigma^k \setminus S_k$ in which the ith element (from left to right) is the ith length-k substring occurrence in w that is not in S_k. In other words, $\mathcal{S}(w, S_k)$ is the sequence of non-forbidden length-k fragments of w. This allows us to reformulate SFSS:

Shortest Fully-Sanitized String (SFSS)
Input: A string $w \in \Sigma^n$, an integer $k > 1$, and a set $S_k \subset \Sigma^k$.
Output: A shortest string $y \in \Sigma^*$ such that no $s \in S_k$ occurs in y and $\mathcal{S}(w, S_k)$ is a subsequence of $\mathcal{S}(y, S_k)$; or FAIL if no such y exists.

▶ **Example 6.** Consider again the instance from Example 4. We have $\mathcal{S}(w, S_k) = \texttt{abbb}, \texttt{bbba}, \texttt{bbaa}, \texttt{baaa}, \texttt{aaab}, \texttt{abaa}$. A solution to the SFSS problem is string $y = \texttt{abbbaaabbbabaa}$. We have that $\mathcal{S}(w, S_k)$ is a subsequence of $\mathcal{S}(y, S_k) = \texttt{abbb}, \texttt{bbba}, \texttt{bbaa}, \texttt{baaa}, \texttt{aaab}, \texttt{aabb}, \texttt{abbb}, \texttt{bbba}, \texttt{bbab}, \texttt{baba}, \texttt{abaa}$. Moreover, no $s \in S_k$ occurs in y and y is a shortest such string.

A solution y to the SFSS problem has the following attractive properties, which are related to privacy or utility: (i) no forbidden substring occurs in y (privacy); (ii) y has as a subsequence the sequence of non-forbidden length-k substrings of w (utility); and (iii) y is the shortest possible (utility).

In Section 5, we reduce the SFSS problem to $d \leq n$ special instances of the SSFS problem. Each such special instance can be seen as seeking for a shortest path in the complete de Bruijn graph of order k over Σ [22] in the presence of forbidden edges. The *complete de Bruijn graph* of order k over an alphabet Σ is a directed graph $G_k = (V_k, E_k)$, where the set of nodes $V_k = \Sigma^{k-1}$ is the set of length-$(k-1)$ strings over Σ. There is an edge $(u, v) \in E_k$ if and only if the length-$(k-2)$ suffix of u is the length-$(k-2)$ prefix of v. There is therefore a natural correspondence between an edge (u, v) and the length-k string $u[1]u[2]\ldots u[k-1]v[k-1]$. We will thus sometimes abuse notation and write $S_k \subset E_k$. In particular, one such instance asks for constructing a shortest path from node u to node v, with $u, v \in V_k$, avoiding edges $S_k \subset E_k$. We thus finally apply Theorem 3, with $k - 1 = |u| = |v|$ and $S = S_k$, $d \leq n$ times to obtain Theorem 7.

▶ **Theorem 7.** *Given a string w of length n over an alphabet Σ, an integer $k > 1$, and a set $S_k \subset \Sigma^k$, SFSS can be solved in $\mathcal{O}(nk|S_k| \cdot |\Sigma|)$ time.*

Other Related Work. Graph reachability is a classic problem in computer science [34, 39, 19, 40]. It refers to the ability to get from one node to another within a graph. In particular, computing shortest paths is one of the most well-studied algorithmic problems. Graph reachability and shortest path computation in the presence of failing nodes or of failing edges becomes a much more challenging task [14, 7, 6, 4, 17, 5, 16, 3, 32]. One obvious, yet important, application of accommodating such failures is in geographic routing [36].

To the best of our knowledge, reachability and shortest path computation in complete de Bruijn graphs in the presence of failing edges has not been considered before. Theorem 2 and Theorem 3 directly solve the reachability and shortest path versions, respectively. Interestingly, Theorem 3 constructs a shortest path in the presence of arbitrarily-long *failing paths*. Our results, other than in data sanitization, may thus be of independent interest.

2 Algorithmic Toolkit

Let M be a finite nonempty set of strings over Σ of total length m. The *trie* of M, denoted by $\mathsf{TR}(M)$, is a deterministic finite automaton that recognizes M with the following features [20]. Its set of states (nodes) is the set of prefixes of the elements of M; the initial state (root node) is ε; the set of terminal states (leaf nodes) is M; and edges are of the form $(u, \alpha, u\alpha)$, where u and $u\alpha$ are nodes and $\alpha \in \Sigma$. The size of $\mathsf{TR}(M)$ is thus $\mathcal{O}(m)$. The *compacted trie* of M, denoted by $\mathsf{CT}(M)$, contains the root node, the branching nodes, and the leaf nodes of $\mathsf{TR}(M)$. The term compacted refers to the fact that $\mathsf{CT}(M)$ reduces the number of

nodes by replacing each maximal branchless path segment with a single edge, and it uses a fragment of a string $s \in M$ to represent the label of this edge in $\mathcal{O}(1)$ machine words. The size of $\mathsf{CT}(M)$ is thus $\mathcal{O}(|M|)$. When M is the set of suffixes of a string y, then $\mathsf{CT}(M)$ is called the *suffix tree* of y, and we denote it by $\mathsf{ST}(y)$. The suffix tree of a string of length n over an alphabet $\Sigma = \{1, \ldots, n^{\mathcal{O}(1)}\}$ can be constructed in $\mathcal{O}(n)$ time [26]. The *generalized suffix tree* of strings $y_1 \ldots, y_k$ over Σ, denoted by $\mathsf{GST}(y_1, \ldots, y_k)$, is the suffix tree of string $y_1\$_1 \ldots y_k\$_k$, where $\$_1, \ldots, \$_k$ are distinct letters not from Σ.

We next recall some basic concepts on randomized algorithms. For an input of size n and an arbitrarily large constant c fixed prior to the execution of a randomized algorithm, the term *with high probability* (whp), or inverse-polynomial probability, means with probability at least $1 - n^{-c}$. When we say that the time complexity of an algorithm holds with high probability, it means that the algorithm terminates in the claimed complexities with probability $1 - n^{-c}$. Such an algorithm is referred to as *Las Vegas whp*. When we say that an algorithm returns a correct answer with high probability, it means that the algorithm returns a correct answer with probability $1 - n^{-c}$. Such an algorithm is referred to as *Monte Carlo whp*.

A *static dictionary* is a data structure that maintains a set K of items that are known in advance. Each item may be associated with some satellite information. A *perfect hash function* for a set K is a hash function that maps the items in K to a set of integers with no collisions. There exists a Las Vegas whp algorithm that constructs a linear-sized static dictionary to maintain S that employs a perfect hash function and supports look-up queries in constant time per query [27]. A *dynamic dictionary* is a data structure that maintains a dynamic set K of items; i.e., a set of items that are not known in advance. Each item may be associated with some satellite information. There exists a Monte Carlo whp algorithm that constructs a linear-sized dynamic dictionary to maintain K that employs a perfect hash function dynamically and supports insert and look-up queries in constant time per query [24].

The *Karp-Rabin fingerpint* (KRF) of a string y over an integer alphabet is defined as $\phi_{q,r}(y) = \sum_{i=1}^{|y|} y[i]r^{|y|-i} \mod q$, where q is a prime number and r is a random integer in $[1, q]$ [35]. A crucial property of KRFs is that, with high probability, no collision occurs among the length-k substrings of a given string. To see this, consider two strings $s \neq t$ each of length k. The polynomial $\phi_{q,r}(s) - \phi_{q,r}(t)$ has at most k roots modulo (prime) q, so the two strings collide, i.e., $\phi_{q,r}(s) = \phi_{q,r}(t)$, with probability no more than $\frac{k}{q-1}$. Thus, for sufficiently large q, we can avoid all possible collisions between the length-k substrings of a string of length n. In particular, for a sufficiently large prime q such that $\log q \in \Theta(\log n)$, $\phi_{q,r}$ is collision-free over all length-k substrings of any fixed string y of length n with high probability. If, however, n is not known in advance, we take $\log q \in \Theta(w)$ instead, where w is the machine word size. We thus work in the word RAM model, where the word size always satisfies $w = \Omega(\log n)$, for any input of size n. We also assume that all standard arithmetic operations between $\mathcal{O}(w)$-bits integers take constant time. The following result is known.

▶ **Lemma 8** ([35]). *For any strings a and b, if we are given $\phi_{q,r}(a)$ and $\phi_{q,r}(b)$, then $\phi_{q,r}(ab)$ can be computed in $\mathcal{O}(1)$ time. If we are given $\phi_{q,r}(ab)$ and $\phi_{q,r}(a)$, then $\phi_{q,r}(b)$ can be computed in $\mathcal{O}(1)$ time.*

Proof. For any a and b, we have $\phi_{q,r}(ab) = (\phi_{q,r}(a)r^{|b|} + \phi_{q,r}(b)) \mod q$ and $\phi_{q,r}(b) = (\phi_{q,r}(ab) - \phi_{q,r}(a)r^{|b|}) \mod q$ by the rules of modular arithmetic. Thus we can compute $\phi_{q,r}(ab)$ from $\phi_{q,r}(a)$ and $\phi_{q,r}(b)$, and $\phi_{q,r}(b)$ from $\phi_{q,r}(ab)$ and $\phi_{q,r}(a)$ in $\mathcal{O}(1)$ time. ◀

3 String Existence Avoiding Forbidden Length-k Substrings

In this section we solve the SEFS problem: is there a string $x \in \Sigma^*$ such that u is a prefix of x, v is a suffix of x, and no $s \in S_k$ occurs in x, where $u, v \in \Sigma^*$ and S_k is a subset of Σ^k?

We start by showing how to solve SEFS efficiently when the strings u, v are of fixed length $k-1$. We will later show how to generalize this result to u and v of any length. We follow a graph-theoretic approach by modelling the problem in terms of complete de Bruijn graphs.

Recall that the complete de Bruijn graph of order k over an alphabet Σ is a directed graph $G_k = (V_k, E_k)$ with $V_k = \Sigma^{k-1}$ and $E_k = \{(u, v) \in V_k \times V_k \mid u[1] \cdot v = u \cdot v[k-1]\}$. A *path* in G_k is a finite sequence of elements from E_k, which joins a sequence of elements from V_k. The *de Bruijn sequence* of order k over Σ is a string in which each element of Σ^k occurs as a substring exactly once [33].

By reachability, we refer to a path in G_k, which starts with a fixed *starting node* u, its infix is a sought (possibly empty) *middle path*, and it ends with a fixed *ending node* v. We consider this notion of reachability in G_k in the presence of *forbidden edges* (or *failing edges*) represented by the set S_k of forbidden length-k substrings over alphabet Σ.

We say that a subgraph $G_k^S = (V_k^S, E_k^S)$ of a complete de Bruijn graph G_k *avoids* $S_k \subset \Sigma^k$ if it consists of all nodes of G_k and all edges of G_k but the ones that correspond to the strings in S_k, that is, if $V_k^S = V_k$ and $E_k^S = E_k \setminus \{(u, v) \in E_k \mid u \cdot v[k-1] \in S_k\}$. Given $u, v \in \Sigma^{k-1}$ and $S_k \subset \Sigma^k$, it can be readily verified that there is a bijection between strings in Σ^n with prefix u and suffix v that do not contain any strings in S_k and paths of length $n-k+1$ that start at u and end at v in G_k^S. In what follows, we show that one can quickly decide whether a node v is reachable from a node u (cf. the SEFS problem) by visiting only a limited portion of G_k^S, even though a shortest such path may be very long.

Our result relies on the notion of the *isoperimetric number* (a.k.a. Cheeger constant or conductance) of a graph. The isoperimetric number measures the "bottleneckedness" of a graph, that is, whether there is a way to partition the nodes into two sets such that the number of edges connecting the two is small compared to the size of the smaller set. More formally, given an undirected graph $G^{\mathrm{u}} = (V, E)$ and a subset of nodes $A \subset V$, the *edge boundary* of A is $\partial_{\mathrm{u}} A = \{\{x, y\} \in E \mid x \in A, y \in V \setminus A\}$: in other words, $\partial_{\mathrm{u}} A$ is the cut-set of the cut $(A, V \setminus A)$. The isoperimetric number of G^{u} is then $h_{\mathrm{u}}(G^{\mathrm{u}}) = \min_{1 \le |A| \le \frac{|V|}{2}} \frac{|\partial_{\mathrm{u}} A|}{|A|}$.[2]

Since we consider *directed* de Bruijn graphs, we will make use of the following notion of isoperimetric number, tailored for the directed case: $h(G) = \min_{1 \le |A| \le \frac{|V|}{2}} \frac{|\partial A|}{|A|}$, where $\partial A = \{(x, y) \in E \mid x \in A, y \in V \setminus A\}$ is the *directed edge boundary* of A, that is, the set of edges outgoing from A. The next lemma, which bounds the isoperimetric number $h(G)$ of a complete directed de Bruijn graph G, was given in [25, Lemma 3.5]. The proof is based on an analogous result on undirected de Bruijn graphs by Delorme and Tillich [23], and we report it here for completeness.

▶ **Lemma 9** ([25]). *Let $G_k = (V_k, E_k)$ be the complete de Bruijn graph of order k over an alphabet Σ. Then*

$$h(G_k) = \min_{1 \le |A| \le \frac{|V_k|}{2}} \frac{|\partial A|}{|A|} \ge \frac{|\Sigma|}{4(k-2)}, \tag{1}$$

where $A \subset V_k$ is a subset of nodes and ∂A is the edge boundary of A.

[2] In the literature, the isoperimetric number of G is also often denoted by $\phi(G)$.

Proof. Let $A \subseteq V_k$ be a cut. The nodes of A have exactly $|\Sigma| \cdot |A|$ ingoing and outgoing edges in total (including self-loops), as in a complete de Bruijn graph each node has exactly $|\Sigma|$ ingoing and $|\Sigma|$ outgoing edges. Let $E[A]$ be the set of edges of which both endpoints are in A. Then $|\partial A| = |\Sigma| \cdot |A| - |E[A]| = |\partial(V_k \setminus A)|$. Note that the undirected edge boundary of A is $\partial_{\mathrm{u}} A = \partial A \cup \partial(V_k \setminus A)$, and $|\partial A| = (|\partial A| + |\partial(V_k \setminus A)|)/2 \geq |\partial A \cup \partial(V_k \setminus A)|/2$. Therefore

$$h(G_k) = \min_{1 \leq |A| \leq \frac{|V_k|}{2}} \frac{|\partial A|}{|A|} \geq \frac{1}{2} \min_{1 \leq |A| \leq \frac{|V_k|}{2}} \frac{|\partial A \cup \partial(V_k \setminus A)|}{|A|} = h_{\mathrm{u}}(G_k^{\mathrm{u}})/2 \geq \frac{|\Sigma|}{4(k-2)},$$

where $h_{\mathrm{u}}(G_k^{\mathrm{u}})$ is the isoperimetric number of the simple, *undirected* version of the complete de Bruijn graph G_k, and the last inequality follows from [23, Theorem 9]. ◀

Main Idea. Lemma 9 states, roughly speaking, that complete de Bruijn graphs have quite a high isoperimetric number, meaning that the edge boundary of any subset of nodes is large compared to the number of nodes in the subset. This implies that eliminating (relatively) few edges from a complete de Bruijn graph is not enough to separate a (relatively) large set of nodes from the rest. Since our goal is to decide whether there exists a path in G_k^S connecting u and v, and S_k has the effect of removing a number of edges from the complete de Bruijn graph G_k, Lemma 9 implies that S_k is large enough to have u and v being separate only if either the set of nodes reachable from u or the set of nodes from which v can be reached is (relatively) small: Figure 1 illustrates this concept.

Our idea is thus to check whether the set of nodes reachable from u or the set of nodes from which v can be reached in G_k^S is small enough to have them separated by S_k or not. If these sets are small enough our algorithm answers NO; otherwise it answers YES. Let us remark that our input consists only of $u \in \Sigma^*$, $v \in \Sigma^*$, $k > 0$ and $S_k \subset \Sigma^k$, and thus G_k^S is not given explicitly: we will generate only the portion of G_k^S that is sufficient to decide SEFS.

The Algorithm

The algorithm relies on the bound given by Lemma 9. Rearranging the factors we get

$$|A| \leq \frac{4(k-2)}{|\Sigma|} |\partial A| \tag{2}$$

for all $A \subset V_k$ with $|A| \leq |V_k|/2$ in the complete de Bruijn graph $G_k = (V_k, E_k)$. Lemma 10 formalizes the main idea and gives a linear-time algorithm for deciding SEFS when $|u| = k-1$ and $|v| = k-1$. We then extend the algorithm for u or v of arbitrary length.

▶ **Lemma 10.** *Given an integer $k > 1$, two strings $u, v \in \Sigma^{k-1}$, and a set $S_k \subset \Sigma^k$, SEFS can be solved in $\mathcal{O}(k|S_k|)$ time.*

Proof. We start by generating the portion of $G_k^S = (V_k^S, E_k^S)$ that can be reached from u in a breadth-first fashion, thus generating an edge only if the corresponding length-k string is not in S_k. We stop when we cannot reach any new nodes or if we have reached $\lfloor \frac{4(k-2)|S_k|}{|\Sigma|} \rfloor + 1$ distinct nodes. Let A be this set of *reachable* nodes from u. If $|A| < \left\lfloor \frac{4(k-2)|S_k|}{|\Sigma|} \right\rfloor + 1$ then A contains all and only the nodes which can be reached from u. In this case there exists a path from u to v if and only if $v \in A$, i.e., v is one of the reachable nodes. Otherwise, we repeat the same procedure, this time using v as the starting point and traversing the edges backwards: let B be set of nodes we reached backwards from v. Again, if $|B| < \lfloor \frac{4(k-2)|S_k|}{|\Sigma|} \rfloor + 1$, then there exists a path from u to v if and only if $u \in B$.

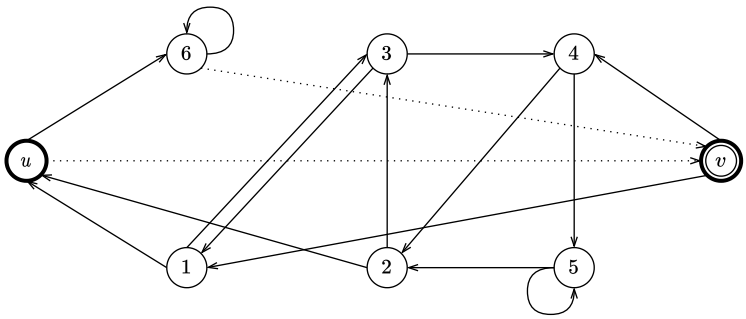

Figure 1 A schematic representation of the complete de Bruijn graph of order 4 over an alphabet of size 2. The dotted edges correspond to a set S_4 of size 2, showing that its size is enough to separate a set of up to two nodes from the rest, and thus, in this example, to make v unreachable from u. It can be easily verified that $|S_4| = 2$ is not enough to separate any 3 nodes from the rest.

Suppose however that $|A| = |B| = \lfloor \frac{4(k-2)|S_k|}{|\Sigma|} \rfloor + 1$, and thus A and B do not contain all the nodes that are reachable from u and from which can reach v, respectively. We claim that in this case v is always reachable from u. In fact, if A and B have a nonempty intersection, then clearly there exists a path from u to v. Otherwise, suppose for a contradiction that there does not exist any path from u to v in the whole G_k^S. Let $V_k \supseteq A' \supseteq A$ be set of nodes which are reachable from u in the whole G_k^S and let $V_k \supseteq B' \supseteq B$ be the set of nodes which can reach v. Since there is no path from u to v in G_k^S, the sets A' and B' are disjoint. Therefore one of them contains at most half the nodes, that is, $\lfloor \frac{4(k-2)|S_k|}{|\Sigma|} \rfloor + 1 \leq \min\{|A'|, |B'|\} \leq |V_k|/2$. Applying Equation (2) to the smallest of the two, we get $|\partial A'| > |S_k|$ or $|\partial B'| > |S_k|$. However note that, since there does not exist a path from u to v in G_k^S, $\partial A' \cup \partial B'^C \subseteq E_k \setminus E_k^S$, where $\partial B'^C$ denotes the directed edge boundary of the complement of B' in V_k. Since in a complete de Bruijn graph each node has the same number of incoming and outgoing edges, it holds $|\partial B'| = |\partial B'^C|$. This is a contradiction. Therefore, we conclude that there exits a path from u to v whenever $|A| = |B| = \lfloor \frac{4(k-2)|S_k|}{|\Sigma|} \rfloor + 1$.

Now that we have bounded the total number of nodes of G_k^S that are needed to decide SEFS, let us describe how we can generate them efficiently. We start by computing the KRF of every forbidden substring, and maintain them in a static dictionary $\mathsf{F}(S_k)$ [27]. This can be done in $\mathcal{O}(k|S_k|)$ time. During the whole process, we also maintain a dynamic dictionary GN [24], where we insert the KRFs of the generated nodes. From each generated node $w = w[1\,..\,k-1]$, we thus need to (i) compute the KRF of string $w\alpha$ corresponding to an outgoing edge in the complete de Bruijn graph G_k, for all $\alpha \in \Sigma$; (ii) check whether the KRF of string $w\alpha$, for all $\alpha \in \Sigma$, is in $\mathsf{F}(S_k)$; (iii) check which of the non-forbidden edges $w\alpha$ lead to a node $w[2\,..\,k-1]\alpha$ whose KRF is not in GN; and (iv) update GN by adding the KRFs of the latter nodes $w[2\,..\,k-1]\alpha$ to GN. Since the portion of G_k^S that we generate is connected, the strings of which we compute the KRFs at any step of this procedure can be obtained by appending α to w, and then by chopping off the first letter of w to obtain $w[2\,..\,k]\alpha$, for all $\alpha \in \Sigma$. By using Lemma 8, we can compute each such KRF in $\mathcal{O}(1)$ time per α, except for the first node, where we spend $\mathcal{O}(k)$ time. Since in G_k there are $|\Sigma|$ edges outgoing from each node, we can compute all KRFs at w and look them up in $\mathsf{F}(S_k)$ and in GN in $\mathcal{O}(|\Sigma|)$ time in total. The new nodes can be inserted in GN in $\mathcal{O}(|\Sigma|)$ time in total as well.

Since we generate $\mathcal{O}(k|S_k|/|\Sigma|)$ nodes in total, and since we spend $\mathcal{O}(|\Sigma|)$ time to process each such node, the overall time required by the above algorithm is $\mathcal{O}(k|S_k|)$. ◀

Let us now discuss the case where u or v are not of fixed length $k-1$: note that we still consider forbidden substrings of fixed length k, which determines the order of G_k. In particular, we consider the case where u or v are of length smaller than $k-1$ (Case 1) and the case where u or v are longer than $k-1$ (Case 2). We finally combine all possible cases.

Case 1: $|u| < k-1$ or $|v| < k-1$. When $|u| < k-1$ (resp. $|v| < k-1$) we should add to the set of reachable nodes A (resp. to B) all the nodes that have u as a suffix (resp. v as a prefix), and then generate the portion of G_k^S reachable from each of them (resp. that can reach each of them), adding the new reached nodes to A (resp. to B) until either its size exceeds the bound given by Lemma 9 or we cannot find any new nodes, again as we described in the proof of Lemma 10.

To show that this can be done efficiently, let us start by observing that the number of nodes from which we start generating the graph increases when the length of u (resp. of v) decreases. As a consequence, for u or v short enough we can decide the problem in constant time. Indeed, if $\max(|u|,|v|) < (k-1) - \log_{|\Sigma|}\left(\frac{4(k-1)|S_k|}{|\Sigma|}\right)$, then the answer is always positive. This is because of a simple counting argument: let us focus on u, as the argument for v is the same. The number of nodes of G_k of which u is a suffix is $|\Sigma|^{(k-1)-|u|}$, which is greater than $\frac{4(k-1)|S_k|}{|\Sigma|}$ whenever $|u| < (k-1) - \log_{|\Sigma|}\left(\frac{4(k-1)|S_k|}{|\Sigma|}\right)$, implying the existence of a path from u to v.

We thus only need to consider the case where u (resp. v) is of length between $(k-1) - \log_{|\Sigma|}\left(\frac{4(k-1)|S_k|}{|\Sigma|}\right)$ and $k-2$. Let us focus on u, as the procedure for v is entirely analogous. Let $d = (k-1) - |u|$. The starting nodes that we need to generate are all and only the length-$(k-1)$ strings over Σ of the form pu, where p is one of the $|\Sigma|^d$ strings of length $k-1-|u|$ over Σ. To obtain them and compute their KRFs efficiently we proceed as follows. We first compute the KRF of u in $\mathcal{O}(|u|) = \mathcal{O}(k)$ time. We then construct the de Bruijn sequence of order d over Σ in time $\mathcal{O}(|\Sigma|^d)$ [33], which is in $\mathcal{O}(k|S_k|/|\Sigma|)$ as we are considering $|u| > (k-1) - \log_{|\Sigma|}\left(\frac{4(k-1)|S_k|}{|\Sigma|}\right)$. We then use a sliding window of size d over the de Bruijn sequence to compute the KRFs of its length-d fragments in overall $\mathcal{O}(|\Sigma|^d) = \mathcal{O}(k|S_k|/|\Sigma|)$ time using Lemma 8. For each length-d fragment p, we apply Lemma 8 again to compute the KRF of node pu in $\mathcal{O}(1)$ time from the KRFs of u and p. Therefore the whole algorithm takes $\mathcal{O}(k|S_k|/|\Sigma|)$ time to generate the starting nodes and compute their KRFs, plus $\mathcal{O}(k|S_k|)$ time to apply the algorithm described in the proof of Lemma 10 from these starting nodes.

Case 2: $|u| > k-1$ or $|v| > k-1$. When $|u| > k-1$ (resp. $|v| > k-1$) it suffices to construct the path which spells u (resp. v) in G_k^S and run the algorithm described in the proof of Lemma 10 from the last node of the path (resp. from the first node). While constructing the path, we compute the KRFs of the length-k substrings of u (resp. v) and check on the dictionary of forbidden substrings whether they are forbidden. If any substring of u (resp. v) is forbidden, then the answer to SEFS is clearly NO. This process requires $\mathcal{O}(|u|)$ (resp. $\mathcal{O}(|v|)$) time in addition to the time required by Lemma 10.

The correctness of the algorithm follows by Lemma 9. Then combining Lemma 10, Case 1, and Case 2 leads to the main result of this section.

▶ **Theorem 2.** *Given an integer $k > 1$, two strings $u, v \in \Sigma^*$, and a set $S_k \subset \Sigma^k$, SEFS can be solved in $\mathcal{O}(|u| + |v| + k|S_k|)$ time.*

▶ Remark 11. The algorithm for obtaining Theorem 2 is Monte Carlo whp due to the use of KRFs and dynamic dictionaries.

Reachability in Complete de Bruijn Graphs

Note that since G_k is complete, it can be specified by k and Σ in $\mathcal{O}(1)$ machine words. Let us now formally define the following reachability problem on complete de Bruijn graphs.

Reachability in de Bruijn Graphs Avoiding Forbidden Edges (RFE) **Input:** The complete de Bruijn graph $G_k = (V_k, E_k)$ of order $k > 1$ over an alphabet Σ, nodes $u, v \in V_k$, and a set $S_k \subset E_k$. **Output:** YES if there exists a path from u to v avoiding any $e \in S_k$; or NO otherwise.

Lemma 10 directly translates to the following corollary.

▶ **Corollary 12.** *Given the complete de Bruijn graph $G_k = (V_k, E_k)$ of order k over an alphabet Σ, nodes $u, v \in V_k$, and a set $S_k \subset E_k$, RFE can be solved in $\mathcal{O}(k|S_k|)$ time.*

4 Shortest String Avoiding Forbidden Substrings

In this section we solve the SSFS problem: construct a shortest $x \in \Sigma^*$ such that u is a prefix of x, v is a suffix of x, and no $s \in S$ occurs in x, where $u, v \in \Sigma^*$ and $S \subset \Sigma^*$.

Let $||S|| = \sum_{s \in S} |s|$. We make the standard assumption that Σ is a subset of $[1, |u| + |v| + ||S|| + 1]$. If this is not the case, we use a static dictionary [27] to do so in $\mathcal{O}(|u| + |v| + ||S||)$ time. Note that all letters of Σ which are neither in u or in v nor in one of the strings in S are interchangeable. They can therefore all be replaced by a single new letter, reducing the alphabet to a size of at most $|u| + |v| + ||S|| + 1$. We will henceforth assume that Σ is such a reduced alphabet. Further note that the input size of the SSFS instance is $(||S|| + |u| + |v|) \log |\Sigma|$ bits or $||S|| + |u| + |v|$ machine words. We further assume that set S is *anti-factorial*, i.e., no $s_1 \in S$ is a proper substring of another element $s_2 \in S$. If that is not the case, we take the set without such s_2 elements to be S. This can be done in $\mathcal{O}(||S||)$ time by constructing the generalized suffix tree of the original S after reducing Σ [26].

Main Idea. We say that a string y is *S-dangerous* if $y = \varepsilon$ or y is a proper prefix of an $s \in S$; we drop S from S-dangerous when this is clear from the context. We aim to construct a labeled directed graph $G(D, E)$ as follows. The set of nodes is the set D of dangerous strings. There exists a directed edge labeled with $\alpha \in \Sigma$ in the set E of edges from node w_1 to node w_2, if $w_1\alpha$ is not in S and w_2 is the longest dangerous suffix of $w_1\alpha$.

Recall that u and v must be a prefix and a suffix of the string x we need to construct, respectively. We set the longest dangerous string in D that is a suffix of u to be the *source* node. We set every node w, such that wv does not contain a string of S, to be a *sink* node. A shortest path from the source node to any sink node corresponds to a shortest such x, where u and v are not allowed to overlap. The overlap case is treated separately.

The Algorithm

The algorithm has two main stages. In the first stage, we construct the graph $G(D, E)$. In the second stage, we find the source and the sinks, and we construct a shortest string x.

Crochemore, Mignosi and Restivo [21] showed how to construct a complete DFA accepting strings over Σ, which do not contain any forbidden substring from S. This is precisely the directed graph $G(D, E)$.[3] We show that this DFA has $\Theta(||S||)$ states and $\Theta(||S|| \cdot |\Sigma|)$ edges

[3] In what follows, we use the term graph and automaton interchangeably, depending on the context.

in the worst case. If we take this DFA and multiply it with the automaton accepting strings of the form uwv, with $w \in \Sigma^*$, we get another automaton accepting all strings of length at least $|u| + |v|$ starting with u, ending with v and not containing any element $s \in S$ as a substring. Since this product automaton would have $\mathcal{O}((|u| + |v|)||S||)$ nodes, we will instead show an efficient way to compute the appropriate source and sink nodes on $G(D, E)$ in $\mathcal{O}(|u|+|v|+||S||)$ time resulting in a total time cost of $\mathcal{O}(|u|+|v|+||S||\cdot|\Sigma|)$ (Theorem 3). We start by showing how $G(D, E)$ can be constructed efficiently for completeness (see also [21]).

Constructing the Graph

First, we construct the trie of the strings in S. This takes $\mathcal{O}(||S||)$ time [20]. We merge the leaf nodes, which correspond to the strings in S, into one forbidden node s'. Note that all other nodes correspond to dangerous strings. We can therefore identify the set of nodes with $D' = D \cup \{s'\}$. We turn this into an automaton by computing a transition function $\delta : D' \times \Sigma \to D'$, which sends each pair $(w, \alpha) \in D \times \Sigma$ to the longest dangerous or forbidden suffix of $w\alpha$ and $(s', \alpha) \in \{s'\} \times \Sigma$ to s'. We can then draw the edges corresponding to the transitions to obtain the graph $G(D, E)$. To help constructing this transition function, we also define a failure function $f : D \to D$ that sends each dangerous string to its longest proper dangerous suffix, which is well-defined because the empty string ε is always dangerous.

In the trie, we already have the edges corresponding to $\delta(w, \alpha) = w\alpha$ if $w\alpha \in D'$. We first add $\delta(s', \alpha) = s'$, for all $\alpha \in \Sigma$. For the failure function, note that $f(\varepsilon) = f(\alpha) = \varepsilon$.

To find the remaining values, we traverse the trie in a breadth first search manner. Let w be an internal node of the trie, that is, a dangerous string of length $\ell > 0$. Then

$$f(w) = \delta(f(w[1\,..\,\ell-1]), w[\ell]) \text{ and } \delta(w, \alpha) = \begin{cases} w\alpha & \text{if } w\alpha \in D' \\ \delta(f(w), \alpha) & \text{if } w\alpha \notin D' \end{cases}.$$

Note that this is well defined, because $w[1\,..\,\ell-1]$ and $f(w)$ are dangerous strings shorter than w, so the corresponding function values are already known.

Once we have computed the transition function and created the corresponding automaton, we delete s' and all its edges thus obtaining $G(D, E)$. To ensure that we can access the node $\delta(w, \alpha)$ in constant time we use a static dictionary on the nodes of $G(D, E)$ [27]. We can alternatively implement the transition functions by arrays in $\Theta(|\Sigma|)$ space per array.

Observe that we need to traverse $|D|$ nodes and compute $|\Sigma| + 1$ function values at each node (one value for f and $|\Sigma|$ values for δ). Every function value is computed in constant time, and therefore the total time complexity of the construction step is $\mathcal{O}(||S|| + |D| \cdot |\Sigma|)$.

▶ **Lemma 13.** *$G(D, E)$ has $\Omega(||S||)$ states and $\Omega(||S|| \cdot |\Sigma|)$ edges in the worst case. $G(D, E)$ can be constructed in the optimal $\mathcal{O}(||S|| \cdot |\Sigma|)$ time.*

Proof. For the first part, consider the instance where S consists of all strings of the form ww with $w \in \Sigma^{k'}$. Then the input size is $\|S\| = 2k'|\Sigma|^{k'}$, while there are more than $k'|\Sigma|^{k'}$ states and $k'|\Sigma|^{k'+1}$ edges. The second part follows from the above discussion (see also [21]). ◀

Constructing a Shortest String

To find the source node, that is, the longest dangerous string that is a suffix of u, we start at the node of $G(D, E)$ that used to be the root of the trie, which corresponds to ε, and follow the edges labeled with the letters of u one by one. This takes $\mathcal{O}(|u|)$ time. Finding the sink nodes directly is more challenging. The trick is to compute the *non-sink nodes* first instead. The non-sink nodes are those dangerous strings $d \in D$ such that the string dv has a

forbidden string $s \in S$ as a substring. To this end, we construct the generalized suffix tree of the strings in S. Recall that this is the compressed trie containing all suffixes of all forbidden strings in S. This takes $\mathcal{O}(||S||)$ time [26]: let us remark that this step has to be done only once. In order to access the children of an explicit suffix tree node by the first letter of their edge label a static dictionary is used [27]. We then find, for each nonempty prefix p of v, all suffixes of all forbidden substrings that are equal to p. We do that by spelling v from the root of the suffix tree of S. There are no more than $||S||$ such suffixes in total, thus the whole process takes $\mathcal{O}(|v| + ||S||)$ time. For each such suffix p, we set the prefix q of the corresponding forbidden substring to be a non-sink node, i.e., we have that $qp \in S$, q is a non-sink, and p is a prefix of v. Recall that all proper prefixes of the elements of S are nodes of $G(D, E)$, and so this is well defined. Any other node is set to be a sink node.

We next consider two cases: $|x| \leq |u| + |v|$ (Case 1) and $|x| \geq |u| + |v|$ (Case 2).

Case 1: $|x| \leq |u| + |v|$. In this case, u and v have a suffix/prefix overlap: a nonempty suffix of u is a prefix of v. We can compute the lengths of all possible suffix/prefix overlaps in $\mathcal{O}(|u| + |v|)$ time and $\mathcal{O}(|u|)$ space by, for instance, constructing the suffix tree of u and spelling the prefixes of v from the root. In order to access the children of an explicit suffix tree node by the first letter of their edge label, a static dictionary is used [27]. We still have to check whether the strings created by such suffix/prefix overlaps contain any forbidden substrings. We do that by starting at ε in $G(D, E)$ and following the edges corresponding to u one by one. If we are at a sink node after following i edges and we have that $u[i+1\,..\,|u|] = v[1\,..\,|u| - i]$, then we output $x = u[1\,..\,i]v$ and halt. Processing all these edges takes $\mathcal{O}(|u|)$ time.

Case 2: $|x| \geq |u| + |v|$. Suppose that Case 1 did not return any feasible path. We then use breadth first search on $G(D, E)$ from the source node to the *nearest* sink node to find a path. If we are at a sink node after following a path spelling string h, then we output $x = uhv$ and halt. In the worst case, we traverse the whole $G(D, E)$. It takes $\mathcal{O}(|E|) = \mathcal{O}(|D| \cdot |\Sigma|)$ time.

In case no feasible path is found in $G(D, E)$, we report FAIL.

Correctness

The paths we find in the algorithm correspond exactly to strings p such that $x = pv$ has u as a prefix and x does not contain any forbidden substrings. Since we search for these paths in order of increasing length, the algorithm will find the shortest p and hence the shortest x, if it exists or report FAIL otherwise.

Complexities

Constructing $G(D, E)$ takes $\mathcal{O}(||S|| + |D| \cdot |\Sigma|)$ time (see also [21]). Finding the source and all sink nodes takes $\mathcal{O}(|u| + |v| + ||S||)$ time. Checking Case 1 takes $\mathcal{O}(|u| + |v|)$ time. Checking Case 2 takes $\mathcal{O}(|D| \cdot |\Sigma|)$ time. It should also be clear that the following bound on the size of the output holds: $|x| \leq |u| + |v| + |D|$. The total time complexity of the algorithm is thus

$$\mathcal{O}(||S|| + |u| + |v| + |D| \cdot |\Sigma|) = \mathcal{O}(|u| + |v| + ||S|| \cdot |\Sigma|).$$

▶ Remark 14. By symmetry, we can obtain a time complexity of $\mathcal{O}(||S|| + |u| + |v| + |D_s| \cdot |\Sigma|)$, where D_s is the set including ε and the proper suffixes of forbidden substrings.

The algorithm uses $\mathcal{O}(||S|| \cdot |\Sigma| + |u|)$ working space, which is the space occupied by $G(D, E)$ and the suffix tree of u.

We obtain the main result of this section.

▶ **Theorem 3.** *Given two strings $u, v \in \Sigma^*$, and a set $S \subset \Sigma^*$, SSFS can be solved in $\mathcal{O}(|u| + |v| + ||S|| \cdot |\Sigma|)$ time, where $||S|| = \sum_{s \in S} |s|$, using $\mathcal{O}(|u| + ||S|| \cdot |\Sigma|)$ space.*

▶ Remark 15. The algorithm for obtaining Theorem 3 is Las Vegas whp due to the use of static dictionaries, which is a standard assumption in algorithms with large alphabets. If $|\Sigma|$ is polynomially bounded in the input size, it can be made deterministic at no extra cost.

A Full Example

In Figure 2, we illustrate the difference between the de Bruijn graph perspective and the automaton perspective. Let $u = \texttt{ab}$, $v = \texttt{ca}$, and $S = \{\texttt{bc}\}$. We start at node ε of the automaton. After processing u we are at source node b. The suffix tree of S contains suffixes c and bc. Since c is a prefix of v, the complementary prefix b of the forbidden substring bc is a non-sink and thus ε a sink. Note that u and v do not have any suffix/prefix overlap. Hence we use breadth first search (Case 2). The shortest path from source b to sink ε is a. Therefore $x = \texttt{abaca}$ is a shortest string with prefix u and suffix v not containing any substring from S.

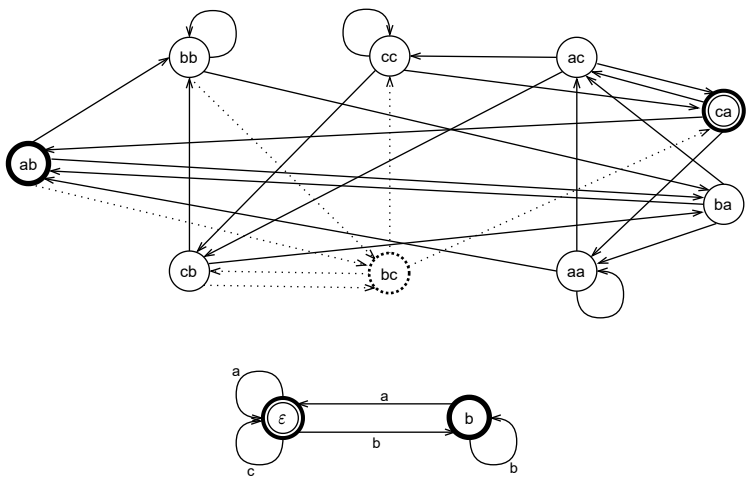

Figure 2 Recall that we have one forbidden substring, bc, of length 2. One could then construct the complete de Bruijn graph of order 3 over alphabet $\Sigma = \{\texttt{a,b,c}\}$ (*Top*); and find the sequence of nodes $\texttt{ab} \to \texttt{ba} \to \texttt{ac} \to \texttt{ca}$, which gives a shortest path starting from node ab, ending at node ca, and avoiding the forbidden node bc. (*Bottom*) the graph $G(D, E)$ after we have computed the source node b and the only sink node ε. The shortest path is then a which gives $x = \texttt{ab} \cdot \texttt{a} \cdot \texttt{ca}$.

Shortest Path in Complete de Bruijn Graphs

We also consider the following more general optimization version of the RFE problem, the reachability problem in complete de Bruijn graphs:

Shortest Path in de Bruijn Graphs Avoiding Forbidden Edges (SPFE) **Input:** The complete de Bruijn graph $G_k = (V_k, E_k)$ of order $k > 1$ over an alphabet Σ, nodes $u, v \in V_k$, and a set $S_k \subset E_k$. **Output:** A shortest path from u to v avoiding any $e \in S_k$; or FAIL if no such path exists.

Note that SPFE is a special case of SSFS. Theorem 3 yields the following corollary.

▶ **Corollary 16.** *Given the complete de Bruijn graph $G_k = (V_k, E_k)$ of order k over an alphabet Σ, nodes $u, v \in V_k$, and a set $S_k \subset E_k$, SPFE can be solved in $\mathcal{O}(k|S_k| \cdot |\Sigma|)$ time.*

5 Shortest Fully-Sanitized String

In this section, we show how to solve the SFSS problem: given $w \in \Sigma^n$ and $S_k \subset \Sigma^k$, construct a shortest $y \in \Sigma^*$ such that no $s \in S_k$ occurs in y and the sequence of non-forbidden length-k substrings of w is a subsequence of the sequence of the length-k substrings of y.

We show that Corollary 16 can be applied on the output of the TFS problem, which we denote by string x, to solve the SFSS problem. Recall from the introduction that x is a shortest string such that no string in S_k occurs in x and the order of all other length-k fragments over Σ is the same in w and in x. In [9], we showed that x is unique and it is always of the form $x = x_0\#_1x_1\#_2\cdots\#_dx_d$, with $x_i \in \Sigma^*$ and $|x_i| \geq k$. It is easy to see why: if we had an occurrence of $\#_ix_i\#_{i+1}$ with $|x_i| \leq k-1$ in x then we could have deleted $\#_ix_i$ to obtain a shorter string x, which is a contradiction. Let us summarize the results related to string x from [9].

▶ **Theorem 17** ([9]). *Let x be a solution to the TFS problem. Then x is unique, it is of the form $x = x_0\#_1x_1\#_2\cdots\#_dx_d$, with $x_i \in \Sigma^*$, $|x_i| \geq k$, and $d \leq n$, it can be constructed in the optimal $\mathcal{O}(n + |x|)$ time, and $|x| = \Theta(nk)$ in the worst case.*

Since $|x_i| \geq k$, each # replacement in x with a letter from Σ can be treated *separately*. In particular, an instance $x_i\#_{i+1}x_{i+1}$ of this problem can be formulated as a shortest path problem in the complete de Bruijn graph of order k over alphabet Σ in the presence of forbidden edges. Corollary 16 can thus be applied d times on $x = x_0\#_1x_1\#_2\cdots\#_dx_d$ to replace the d occurrences of # in x and obtain a final string over Σ: given an instance $x_i\#_{i+1}x_{i+1}$, we set u to be the length-$(k-1)$ suffix of x_i and v to be the length-$(k-1)$ prefix of x_{i+1}. Let us denote by y the string obtained by this algorithm.

▶ **Example 18.** Let $w = \underline{\mathtt{abbbb}}\underline{\mathtt{aaba}}\mathtt{a}$, $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $k = 4$, and $S_k = \{\mathtt{bbbb}, \mathtt{aaba}, \mathtt{abba}\}$ (the instance from Example 4), and the solution $x = \mathtt{abbbaaab\#abaa}$ of the TFS problem. By setting $u = \mathtt{aab}$ and $v = \mathtt{aba}$ in the SPFE problem, we obtain as output the path corresponding to string $p = \mathtt{aabbbaba}$. The prefix aab of p corresponds to the starting node u, its infix bb corresponds to the middle path found, and its suffix aba corresponds to the ending node v. We use p to replace aab#aba and obtain the final string $y = \mathtt{abbbaaabbbabaa}$.

However, to prove that y is a solution to the SFSS problem, we further need to prove that $\mathcal{S}(w, S_k)$ is a subsequence of $\mathcal{S}(y, S_k)$, and that y is a shortest possible such string.

▶ **Lemma 19.** *Let $x = x_0\#_1x_1\#_2\cdots\#_dx_d$, with $x_i \in \Sigma^*$ and $|x_i| \geq k$, be a solution to the TFS problem on a string w, and let y be the string obtained by applying Corollary 16 d times on x to replace the occurrences of #. String y is a shortest string over Σ such that $\mathcal{S}(w, S_k)$ is a subsequence of $\mathcal{S}(y, S_k)$ and no $s \in S_k$ occurs in y.*

Proof. No $s \in S_k$ occurs in y by construction. With a slight abuse of notation, let $\mathcal{S}(x, S_k)$ be the sequence of length-k substrings over Σ occurring in x from left to right. Since x is a solution to the TFS problem, we have that $\mathcal{S}(x, S_k) = \mathcal{S}(w, S_k)$. Since $x_0, x_1, \ldots, x_d$ occur in y in the same order as in x by construction, it follows that $\mathcal{S}(x, S_k) = \mathcal{S}(w, S_k)$ is a subsequence of $\mathcal{S}(y, S_k)$. We now need to show that there does not exist another string y' shorter than y such that $\mathcal{S}(w, S_k)$ is a subsequence of $\mathcal{S}(y', S_k)$ and no $s \in S_k$ occurs in y'. Suppose for a contradiction that such a shorter string y' does exist. Since x is the shortest string such that $\mathcal{S}(x, S_k) = \mathcal{S}(w, S_k)$ and no $s \in S_k$ occurs in x, all the length-k substrings of $x_0, x_1, \ldots, x_d$ are also a subsequence of $\mathcal{S}(y', S_k)$ by hypothesis.

Let $y[\ell_i \mathinner{.\,.} r_i]$ and $y'[\ell'_i \mathinner{.\,.} r'_i]$ be the shortest substrings of y and y', respectively, where the length-k substrings of x_i and x_{i+1} appear and such that $|y[\ell_i \mathinner{.\,.} r_i]| > |y'[\ell'_i \mathinner{.\,.} r'_i]|$ (there must be an i such that this is the case, as we supposed $|y| > |y'|$). Since $y[\ell_i \mathinner{.\,.} r_i]$ is obtained by applying Corollary 16 to the length-$(k-1)$ suffix of x_i and the length-$(k-1)$ prefix of x_{i+1}, it is a shortest string that has x_i as a prefix and x_{i+1} as a suffix, implying that $y'[\ell'_i \mathinner{.\,.} r'_i]$ can only be shorter if it is not of the same form: have x_i as a prefix and x_{i+1} as a suffix. Suppose then that x_i is not a prefix of $y'[\ell'_i \mathinner{.\,.} r'_i]$, and thus there exist two consecutive length-k substrings of x_i that are not consecutive in $y'[\ell'_i \mathinner{.\,.} r'_i]$. But then it is always possible to remove any letters between the two in $y'[\ell'_i \mathinner{.\,.} r'_i]$ to make them consecutive and obtain a string shorter than $y'[\ell'_i \mathinner{.\,.} r'_i]$. This operation does not introduce any occurrences of some $s \in S_k$, as the two length-k substrings are consecutive in x_i which, in turn, does not contain any $s \in S_k$. By repeating this reasoning on any two length-k substrings of x_i and x_{i+1}, we obtain a string y''_i that has x_i as a prefix, x_{i+1} as a suffix and such that $|y''_i| < |y'[\ell'_i \mathinner{.\,.} r'_i]| < |y[\ell_i \mathinner{.\,.} r_i]|$. This is a contradiction, as $y[\ell_i \mathinner{.\,.} r_i]$ is a shortest string that has x_i as a prefix and x_{i+1} as a suffix. ◀

By Theorem 17, Corollary 16, and Lemma 19 we obtain the main result of this section.

▶ **Theorem 7.** *Given a string w of length n over an alphabet Σ, an integer $k > 1$, and a set $S_k \subset \Sigma^k$, SFSS can be solved in $\mathcal{O}(nk|S_k| \cdot |\Sigma|)$ time.*

▶ Remark 20. The algorithm for obtaining Theorem 7 is Las Vegas whp due to the use of Corollary 16 (which relies on Theorem 3). If $|\Sigma|$ is polynomially bounded in the size of the input, the algorithm can be made deterministic at no extra cost.

We stress that the fact that y is the shortest possible is important for utility. Let $G(x)[v]$ denote the total number of occurrences of string v in string x. The *k-gram profile* of x is the vector $G_k(x) = (G(x)[v]), v \in \Sigma^k$. The *$k$-gram distance* between two strings is defined as the L_1-norm of the difference of their k-gram profiles. The k-gram distance is a pseudo-metric that is widely used (especially in bioinformatics), because it can be computed in linear time in the sum of the lengths of the two strings [41]. It is now straightforward to see that the k-gram distance between strings w (input of SFSS) and y (output of SFSS), such that $\mathcal{S}(w, S_k)$ is a subsequence of $\mathcal{S}(y, S_k)$ and no $s \in S_k$ occurs in y, is minimal. Thus, conceptually, SFSS introduces in y the least amount of spurious information.

6 Open Questions

We leave the following questions unanswered:

1. Can the SEFS problem be solved deterministically in $\mathcal{O}(|u| + |v| + k|S_k|)$ time? One could investigate whether the use of KRFs and dynamic dictionaries can be avoided.

2. Can the SSFS problem be solved faster than $\mathcal{O}(|u| + |v| + ||S|| \cdot |\Sigma|)$ time? One should perhaps design a fundamentally different technique that avoids the DFA construction, because, as we have shown, the latter has $\Omega(||S||)$ states and $\Omega(||S|| \cdot |\Sigma|)$ edges.
3. Sometimes we may want to solve many instances of the SSFS problem having the same set of forbidden substrings but different u and v; for example, in the SFSS problem (see Section 5). Can we solve q such instances faster than applying Theorem 3 q times?

References

1 Osman Abul, Francesco Bonchi, and Fosca Giannotti. Hiding sequential and spatiotemporal patterns. *IEEE Trans. Knowl. Data Eng.*, 22(12):1709–1723, 2010. doi:10.1109/TKDE.2009.213.

2 Osman Abul and Harun Gökçe. Knowledge hiding from tree and graph databases. *Data Knowl. Eng.*, 72:148–171, 2012. doi:10.1016/j.datak.2011.10.002.

3 Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single-source fault tolerant shortest path. *ACM Trans. Algorithms*, 16(4):44:1–44:22, 2020. doi:10.1145/3397532.

4 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant reachability for directed graphs. In *DISC*, pages 528–543, 2015. doi:10.1007/978-3-662-48653-5_35.

5 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault-tolerant subgraph for single-source reachability: General and optimal. *SIAM J. Comput.*, 47(1):80–95, 2018. doi:10.1137/16M1087643.

6 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013. doi:10.1007/s00453-012-9621-y.

7 Surender Baswana, Utkarsh Lath, and Anuradha S. Mehta. Single source distance oracle for planar digraphs avoiding a failed node or link. In *SODA*, pages 223–232, 2012. doi:10.1137/1.9781611973099.20.

8 Marie-Pierre Béal, Maxime Crochemore, Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. Computing forbidden words of regular languages. *Fundam. Informaticae*, 56(1-2):121–135, 2003. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi56-1-2-08.

9 Giulia Bernardini, Huiping Chen, Alessio Conte, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. String sanitization: A combinatorial approach. In *ECML PKDD*, volume 11906, pages 627–644, 2019. doi:10.1007/978-3-030-46150-8_37.

10 Giulia Bernardini, Huiping Chen, Alessio Conte, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, Giovanna Rosone, and Michelle Sweering. Combinatorial algorithms for string sanitization. *ACM Trans. Knowl. Discov. Data*, 15(1):8:1–8:34, 2020. doi:10.1145/3418683.

11 Giulia Bernardini, Huiping Chen, Grigorios Loukides, Nadia Pisanti, Solon P. Pissis, Leen Stougie, and Michelle Sweering. String sanitization under edit distance. In *CPM*, pages 7:1–7:14, 2020. doi:10.4230/LIPIcs.CPM.2020.7.

12 Giulia Bernardini, Alessio Conte, Garance Gourdel, Roberto Grossi, Grigorios Loukides, Nadia Pisanti, Solon Pissis, Giulia Punzi, Leen Stougie, and Michelle Sweering. Hide and mine in strings: Hardness and algorithms. In *ICDM*, pages 924–929, 2020. doi:10.1109/ICDM50108.2020.00103.

13 Luca Bonomi, Liyue Fan, and Hongxia Jin. An information-theoretic approach to individual sequential data sanitization. In *WSDM*, pages 337–346, 2016. doi:10.1145/2835776.2835828.

14 Andrei Z. Broder, Danny Dolev, Michael J. Fischer, and Barbara Simons. Efficient fault-tolerant routings in networks. *Inf. Comput.*, 75(1):52–64, 1987. doi:10.1016/0890-5401(87)90063-0.

15 Pascal Caron. Families of locally testable languages. *Theoretical Computer Science*, 242(1):361–376, 2000. doi:10.1016/S0304-3975(98)00332-6.

16 Panagiotis Charalampopoulos, Shay Mozes, and Benjamin Tebeka. Exact distance oracles for planar graphs with failing vertices. In *SODA*, pages 2110–2123, 2019. doi:10.1137/1.9781611975482.127.

17 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *ICALP*, pages 130:1–130:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.130.

18 Chris Clifton and Don Marks. Security and privacy implications of data mining. In *SIGMOD*, pages 15–19, 1996.

19 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

20 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

21 Maxime Crochemore, Filippo Mignosi, and Antonio Restivo. Automata and forbidden words. *Inf. Process. Lett.*, 67(3):111–117, 1998. doi:10.1016/S0020-0190(98)00104-5.

22 Nicolaas Govert de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie V. Wetenschappen*, 49:758–764, 1946.

23 C. Delorme and J.-P. Tillich. The spectrum of de Bruijn and Kautz graphs. *European Journal of Combinatorics*, 19(3):307–319, 1998. doi:10.1006/eujc.1997.0183.

24 Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *ICALP*, pages 6–19, 1990. doi:10.1007/BFb0032018.

25 Igor Dolinka. On free spectra of locally testable semigroup varieties. *Glasgow Mathematical Journal*, 53(3):623–629, 2011. doi:10.1017/S0017089511000188.

26 Martin Farach. Optimal suffix tree construction with large alphabets. In *FOCS*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

27 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

28 Aris Gkoulalas-Divanis and Grigorios Loukides. Revisiting sequential pattern hiding to enhance utility. In *SIGKDD*, pages 1316–1324, 2011. doi:10.1145/2020408.2020605.

29 Aris Gkoulalas-Divanis and Vassilios S. Verykios. An integer programming approach for frequent itemset hiding. In *CIKM*, pages 748–757, 2006. doi:10.1145/1183614.1183721.

30 Aris Gkoulalas-Divanis and Vassilios S. Verykios. Exact knowledge hiding through database extension. *IEEE Trans. Knowl. Data Eng.*, 21(5):699–713, 2009. doi:10.1109/TKDE.2008.199.

31 Robert Gwadera, Aris Gkoulalas-Divanis, and Grigorios Loukides. Permutation-based sequential pattern hiding. In *ICDM*, pages 241–250, 2013. doi:10.1109/ICDM.2013.57.

32 Giuseppe F. Italiano, Adam Karczmarz, and Nikos Parotsidis. Planar reachability under single vertex or edge failures, 2021. doi:10.1137/1.9781611976465.163.

33 L.Ro Ford Jr. A cyclic arrangement of m-tuples. Technical Report Report P-1071, Rand Corporation, 1957.

34 Tiko Kameda. On the vector representation of the reachability in planar directed graphs. *Inf. Process. Lett.*, 3(3):75–77, 1975. doi:10.1016/0020-0190(75)90019-8.

35 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

36 Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Geographic routing made practical. In *NSDI*, 2005. URL: http://www.usenix.org/events/nsdi05/tech/kim.html.

37 Grigorios Loukides and Robert Gwadera. Optimal event sequence sanitization. In *ICDM*, pages 775–783, 2015. doi:10.1137/1.9781611974010.87.

38 Stanley R. M. Oliveira and Osmar R. Zaïane. Protecting sensitive knowledge by data sanitization. In *ICDM*, pages 613–616, 2003. doi:10.1109/ICDM.2003.1250990.

39 Steven Skiena. *The Algorithm Design Manual, Third Edition*. Texts in Computer Science. Springer, 2020. doi:10.1007/978-3-030-54256-6.

40 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004. doi:10.1145/1039488.1039493.

41 Esko Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992. doi:10.1016/0304-3975(92)90143-4.

42 Vassilios S. Verykios, Ahmed K. Elmagarmid, Elisa Bertino, Yücel Saygin, and Elena Dasseni. Association rule hiding. *IEEE Trans. Knowl. Data Eng.*, 16(4):434–447, 2004. doi:10.1109/TKDE.2004.1269668.

43 Yi-Hung Wu, Chia-Ming Chiang, and Arbee L. P. Chen. Hiding sensitive association rules with limited side effects. *IEEE Trans. Knowl. Data Eng.*, 19(1):29–42, 2007. doi:10.1109/TKDE.2007.250583.