Hardness Results on Characteristics for Elastic-Degenerate Strings

Dominik Köppl
University of Yamanashi, Kofu, Japan

Jannik Olbrich
Ulm University, Germany

Abstract

Generalizations of plain strings have been proposed as a compact way to represent a collection of nearly identical sequences or to express uncertainty at specific text positions by enumerating all possibilities. While a plain string stores a character at each of its positions, generalizations consider a set of characters (indeterminate strings), a set of strings of equal length (*generalized degenerate strings*, or shortly *GD strings*), or a set of strings of arbitrary lengths (*elastic-degenerate strings*, or shortly *ED strings*). These generalizations are of importance to compactly represent such type of data, and find applications in bioinformatics for representing and maintaining a set of genetic sequences of the same taxonomy or a multiple sequence alignment. To be of use, attention has been drawn to answering various query types such as pattern matching or measuring similarity of ED strings by generalizing techniques known to plain strings. However, for some types of queries, it has been shown that a generalization of a polynomial-time solvable query on classic strings becomes NP-hard on ED strings, e.g. [Russo et al., 2022]. In that light, we wonder about other types of queries that are of particular interest to bioinformatics: unique substrings, absent words, anti-powers, longest previous factors, and Lempel–Ziv-like compression schemes. While we obtain a polynomial time algorithm for a variation of longest previous factors, we show that all other problems are NP-hard to compute, some of them even under the restriction that the input can be modeled as an indeterminate or GD string.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Elastic-degenerate strings, NP-hardness, longest common factor, minimal unique substring, minimal absent word, anti-power, longest previous factor

Digital Object Identifier 10.4230/LIPIcs.CPM.2026.14

Related Version *Full Version*: `https://arxiv.org/abs/2411.10653`
Workshop Version I: `https://ipsj.ixsq.nii.ac.jp/records/241892`
Workshop Version II: `https://ipsj.ixsq.nii.ac.jp/records/241031`

Supplementary Material
Software (Source Code): `https://github.com/koeppl/edstringcharacteristics`
archived at `swh:1:dir:dd66d0b103bc6c7bbe00497eda1474ab3f19e35b`

Funding *Dominik Köppl*: JSPS KAKENHI Grant Number 25K21150 and Yamanashi Wakate Grant Number 2291.

1 Introduction

In many applications, uncertainty is a known problem that hinders the adaptation of classic algorithms requiring full information. We here model uncertainty by explicitly stating all possibilities. Such a model found popularity when working with biological data such as gene sequences, for which indeterminate strings [96] have been proposed. An indeterminate string models a string that can have multiple alternative characters at any of its positions; each element of an indeterminate string becomes a subset of the input alphabet, which

we call a *symbol*. Symbols generalize the IUPAC notation [71] to represent the ambiguous specification of nucleic acids in DNA or RNA sequences. A generalization of indeterminate strings are generalized degenerate (GD) strings that store strings of equal length instead of characters. Finally, [70] removed the restriction on equal length and called the modelled string an elastic-degenerate (ED) string, which was proposed as a model of a set of similar sequences. From a language theoretic point of view, an ED-string is a regular expression without nested parentheses or Kleene star. Figure 1 gives an example of the three models.

$$\widetilde{T_1} = \mathtt{A}\left\{\begin{matrix}\mathtt{A}\\ \mathtt{C}\end{matrix}\right\}\mathtt{C}\left\{\begin{matrix}\mathtt{G}\\ \mathtt{T}\end{matrix}\right\} \qquad \widetilde{T_2} = \mathtt{A}\left\{\begin{matrix}\mathtt{AG}\\ \mathtt{CA}\end{matrix}\right\}\mathtt{C}\left\{\begin{matrix}\mathtt{G}\\ \mathtt{T}\end{matrix}\right\} \qquad \widetilde{T_3} = \mathtt{A}\left\{\begin{matrix}\mathtt{AG}\\ \varepsilon\end{matrix}\right\}\mathtt{C}\left\{\begin{matrix}\mathtt{G}\\ \mathtt{T}\\ \mathtt{CCT}\end{matrix}\right\}$$

$\mathcal{L}(\widetilde{T_1}) = \{$ AACG, AACT, ACCG, ACCT $\}$

$\mathcal{L}(\widetilde{T_2}) = \{$ AAGCG, AAGCT, ACACG, ACACT $\}$

$\mathcal{L}(\widetilde{T_3}) = \{$ AAGCG, AAGCT, AAGCCCT, ACG, ACT, ACCCT $\}$

Figure 1 Example for an indeterminate string $\widetilde{T_1}$ (left), a GD string $\widetilde{T_2}$ (middle), and an ED string $\widetilde{T_3}$ (right). The language $\mathcal{L}(\widetilde{T_i})$ of each string $\widetilde{T_i}$ is depicted below; the language is the set of strings represented by the indeterminate/GD/ED string. A symbol storing only one character c is written as $\left\{ c \right\}$ or just c.

ED strings found application to encode the consensus of a population of sequences [9, 29, 75] in a multiple sequence alignment. While advances in pattern matching on ED strings spawned recent research interest, not much is known about the computational complexity of typical string characteristics used, among others, in bioinformatics on classic strings.

Our contribution. We here prove NP-hardness for the adaptation of five string problems to ED strings, which makes it unlikely to come up with efficient solutions for these problems on large ED strings.

1. Finding a unique substring of length k in an indeterminate string is NP-hard (Thm. 1).
2. Finding an absent word of length k in an indeterminate string is NP-hard (Thm. 3).
3. Finding a k-anti-power in a GD string is NP-hard (Thm. 5).
4. Computing the longest previous factor length in an ED string is NP-hard (Thm. 7).
5. Covering an indeterminate string with Lempel–Ziv factors is NP-hard (Thms. 9 and 11).

The studied NP-hard problems are not only NP-hard, but also NP-complete. To see that, we show that we can verify a certificate in polynomial time. In all problems except the last, a certificate is a classic string X that solves the addressed problem. To verify that X is a solution, we apply a pattern matching algorithm on the input ED string $\widetilde{S}$, which runs in polynomial time. For instance, a certificate for a unique substring or an absent word is a string X, for which we check that it occurs exactly once or zero times, respectively.

Roadmap. In the remainder of this article, we first give an overview on related work in Section 2. We then give preliminaries in Section 3 and present our hardness results in Sections 4–7. Subsequently, we propose MAX-SAT formulations for finding a minimal unique substring or a minimal absent word in Section 8 and an implementation that can solve both problems in practice on inputs of moderate size. While the conclusion in Section 9 states several open problems, a more elaborated open problem on the inequality testing of two ED strings is described in the appendix in Appendix C. In the appendix, we also give polynomial time algorithms for computing the longest repeating substring in Appendix A and longest previous factors in indeterminate strings in Appendix B.

2 Related Work

We structure the related work in work dedicated to ED strings and work tackling one of the aforementioned problems, however on different types of input (mostly classic strings).

2.1 ED Strings

A big part of research on ED strings has been devoted to pattern matching, structural properties or regularities, the reconstruction of ED strings from indexes that are not necessarily self-indexes, and the comparison of two ED strings.

Pattern Matching. For indeterminate strings, we point out a Boyer–Moore adaptation [64], a combination [96] of ShiftAnd and Boyer–Moore–Sunday, and an KMP-based approach [86]. For ED strings, there is a rather long line on improvements for exact pattern matching [13, 35, 25, 70, 93, 14], with one error [24], or general approximate pattern matching [26, 58, 92]. There are indices for pattern matching on ED texts, based on the suffix tree [59], on the Burrows–Wheeler transform (BWT) [77] or on a graph for read mapping [29]. The first study [59] also gives lower bounds on the time complexity for indexed pattern matching queries. The problem to align a pattern to a GD string [83] or ED string [84] has also recently been studied.

Given that r is the maximum size of the set a symbol in the ED string represents, NP-hardness for order-preserving pattern matching has been shown for $r = 3$ [94] and subsequently even for $r = 2$ [57]. The same article [57] also gives NP-hardness for parameterized pattern matching for $r = 2$.

Structural Properties. For structural properties on indeterminate strings, we are aware of the computation of covers and/or seeds [12, 18], maximal gapped palindromes [11], a study on periodicity [53], an extension of the Lyndon factorization [42], and an augmentation with rank/select data structures [27]. [41] showed that computing the shortest cover is NP-complete, but fixed-parameter tractable (FPT) in the number of symbols with sizes larger than one. Finally, [76] gave a new representation model for indeterminate strings.

Reconstruction. Reconstruction of indeterminate strings from data structures is another active line of research. The reconstruction has been considered from border array, or suffix array and the LCP array [85], from a graph based on the prefix array [7], or from a graph whose vertices are text positions and edges are text positions having matching characters [63]. [34] gave a characterization of whether an array can be the prefix array of a classic string or an indeterminate string. Finally, [28] studied the relation of prefix arrays of indeterminate strings to undirected graphs and border arrays.

Comparison. [9] studied how to compare GD strings. An extension to ED strings is due to [50], where the focus is put on the intersection of the languages of two ED strings, a problem coined as the *ED String Intersection Problem* (EDSI). The authors showed that EDSI is NP-complete if the alphabet is unary while the letters are stored run-length compressed [50, Thm. 9]. Otherwise, EDSI is polynomial-time solvable [50, Thm. 24]. Additionally, they gave conditional lower bounds for computing the intersection.

2.2 Addressed Problems on Classic Strings

We here give motivation for the problems we address in this article by highlighting, for each problem, its importance and existing solutions, which however do not cover ED strings.

Minimal Unique Substring. The problem of computing minimal unique substrings (MUSs) was introduced by Pei et al. [90], and has found application in sequence alignment [4], genome comparison [61], and phylogenetic tree construction [31]. A weaker version, the shortest unique substring (SUS), asks for only local minimality, which can be expressed by a point or region to cover. The interest in computing SUSs resulted in a line of research improvements on the time and space complexities [67, 98, 69, 79, 81]. There are solutions offering a trade-off in both complexities [55, 17]. Also, different settings like the computation within a sliding window [82] or on run-length encoded strings [79] have been proposed. For theoretical guarantees, [80] studied the maximum number of SUSs covering a given query position. Recently, a survey article [3] has been dedicated to SUS computation. Finally, there are variations for computing SUSs within a given range [1, 2] or computing SUSs with k mismatches [65, 95, 8].

Minimal Absent Word. Minimal absent words (MAWs) have been introduced by [91] as biomarkers for potential preventive and curative medical applications. MAWs have been found valuable for phylogeny [33], sequence comparison [37], information retrieval of musical content [36], and the reconstruction of circular binary strings [89]. For computing MAWs, algorithms have been proposed that use the suffix array [19], the directed acyclic word graph (DAWG) [49, 48], or maximal repeats [15]. Algorithms working in parallel [20] or in external memory [62] have also been proposed. Extensions are the computation of MAWs of run-length compressed strings [6], MAWs common to multiple strings [88], or MAWs on trees [44]. Another extension is the computation within a sliding window [38, 82], for which bounds on the number of changes in the answer set based on the sliding movement have been analyzed [5].

Anti-Power. [46] introduced anti-powers as a new combinatorial concept. We are aware of algorithms for computing anti-powers [16, 10], and counting them [73]. Anti-powers have been explicitly analyzed for prefixes of the Thue–Morse word [43, 54] and for aperiodic recurrent words [23]. A specialization of anti-powers is to strengthen inequality by *Abelian inequality*, where all factors of an Abelian anti-power must have pairwise distinct Parikh vectors [45]. Another specialization is due to [30] with the restriction that the computed anti-power must be also a repetition, i.e., an integer power of a word.

Longest-Previous-Factor and LZ77 factorization. The longest previous factor (LPF) array [47, 39] has been proposed for pattern matching, computing the Lempel–Ziv 77 (LZ77) factorization, and detecting runs. There are algorithms [39, 40, 32] computing the LPF array in linear time. These linear-time algorithms directly give linear-time algorithms computing the LZ77 factorization.

3 Preliminaries

We start with introducing basic concepts for strings, and then formalize generalizations to indeterminate, GD, and ED strings.

Strings. Let Σ be an alphabet. An element of Σ^* is called a (classic) *string*. Given a string T, the i-th character of T is denoted by $T[i]$, for an integer $i \in [1..|T|]$, where $|T|$ denotes the length of T. For two integers i and j with $1 \leq i \leq j \leq |T|$, a substring of T starting at position i and ending at position j is denoted by $T[i..j]$, i.e., $T[i..j] = T[i]T[i+1]\cdots T[j]$. A substring P of T with $|P| \geq 1$ is called *proper* if $P \neq T$. We write T^k for the k-power of the string T, i.e., $T^1 = T$ and $T^k = TT^{k-1}$. Further, $\Sigma^k \subset \Sigma^*$ denotes the set of all k-length strings whose characters are from Σ.

Indeterminate and Generalized/Elastic-Degenerate Strings. We study the following extensions of classic strings. For that, we make the distinction between *characters* of the alphabet Σ and *symbols* of a string belonging to one of the three types of generalizations defined as follows. First, an *indeterminate string* is a string $\widetilde{S}[1..n]$ whose symbols are drawn from non-empty subsets of Σ, i.e., $\emptyset \neq \widetilde{S}[i] \subset \Sigma$. Next, a *generalized degenerate string* or *GD string* is a string $\widetilde{S}[1..n]$ whose symbol at the i-th position is a non-empty subset of strings in Σ^{k_i}, i.e., $\emptyset \neq \widetilde{S}[i] \subset \Sigma^{k_i}$ for some k_i depending on i. Finally, an *elastic-degenerate string* or *ED string* is the most general type allowing each symbol to be a set of strings including the empty word ε, which is non-empty and not $\{\varepsilon\}$, i.e., $S[i] \subset \Sigma^*$ with $\emptyset \neq S[i] \neq \{\varepsilon\}$. Each indeterminate string can be expressed as a degenerate string by interpreting characters as strings of length one. For any type of string class, we call its elements *symbols*. Like for strings, we denote with $\widetilde{S}[i..j] = \widetilde{S}[i]\cdots\widetilde{S}[j]$ the ED substring that starts at position i and ends at position j.

The *Cartesian concatenation* of two symbols X and Y is $X \otimes Y := \{xy \mid x \in X, y \in Y\}$. Consider an ED string $\widetilde{S}$ of length n. The *language* of $\widetilde{S}$ is $\mathcal{L}(\widetilde{S}) := \widetilde{S}[1] \otimes \widetilde{S}[2] \otimes \cdots \otimes \widetilde{S}[n]$, cf. [9, Def. 5]. An element of $\mathcal{L}(\widetilde{S})$ has length n if $\widetilde{S}$ is indeterminate, or length $\sum_{i=1}^{n} k_i$ if $\widetilde{S}$ is a GD string. To address a character in an ED string $\widetilde{S}$, we assume that each symbol $\widetilde{S}[i]$ is an ordered list (e.g., ordered canonically in lexicographic order) such that $\widetilde{S}[i][j][k]$ addresses the k-th character of the j-th string in the i-th symbol. We say that (i, j, k) is the *text-position* of $\widetilde{S}[i][j][k]$. Regarding the ED strings of Figure 1, $\widetilde{T_1}[1][1][1] = \mathtt{A}, \widetilde{T_1}[2][2][1] = \mathtt{C}, \widetilde{T_3}[4][3][3] = \mathtt{T}$. The *size* of a symbol is the total length of the characters it contains, where the empty string is counted as having length one, e.g., the sizes of the symbols of $\widetilde{T_3}$ are 1,3,1,5. The *size* of an ED string $\widetilde{S}$, denoted by $||\widetilde{S}||$ is the sum of the sizes of all its symbols. Finally, let $r(\widetilde{S})$ or shortly r denote the maximum number of strings a symbol in $\widetilde{S}$ contains, e.g., the symbols of $\widetilde{T_3}$ contain 1,2,1,3 strings, respectively, so $r(\widetilde{T_3}) = 3$.

We say that a classic string P has an *occurrence* starting at text-position (i, j, k) in an ED string $\widetilde{S}$ if we can factorize P into $P = P_1 \cdots P_m$ such that $\widetilde{S}[i][j][k..] = P_1$, P_m is a prefix of some string in $\widetilde{S}[i+m-1]$ and $P_x \in \widetilde{S}[i+x-1]$ for all $x \in [2..m-1]$. For instance, $P = \mathtt{CAC}$ has an occurrence in $\widetilde{T_2}$ at text-position $(2, 2, 1)$.

4 Hardness of Finding Unique Substrings and Absent Words

In what follows, we reduce 3-SAT with n variables to the decision problem whether a minimal unique substring (MUS), or a minimal absent word (MAW) of length at most n exists. The input of 3-SAT is a formula F in conjunctive normal form (CNF), i.e., F joins a set of clauses C_i by conjunction (AND), where each clause C_i is a disjunction of three literals. We assume that the n variables $x_1, \ldots, x_n$ are ranked.

The idea is to specify for each clause C_i all variable assignments that make C_i unsatisfiable. Subtracting the union of these unsatisfying assignments from all possible assignments gives us all satisfiable assignments. The solution further deviates from hereon for MUSs and

MAWs: For MUSs, if we also express the set of all assignments as an indeterminate substring of $\widetilde{S}$, then the unsatisfiable assignments appear (at least) twice in $\widetilde{S}$ while the satisfying assignments only once. We show that these also coincide with the shortest MUSs if F is satisfiable. For MAWs, if we instead append an indeterminate string to $\widetilde{S}$ covering all possible substrings of length $n-1$, then a MAW must be of length at least n. It is then left to show that a shortest MAW of $\widetilde{S}$ is a substring of length n that represents a satisfying assignment.

4.1 Hardness of Minimal Unique Substring

We start with a formal definition of MUSs based on classic strings. We call a substring U of a classic string T *unique* if it occurs exactly once in T, i.e., there is no other position in T at which a substring starts that matches with U. In particular, we call a unique substring U to be a *minimal unique substring* (*MUS*) of T if every proper substring of U occurs at least twice in T.

We generalize the notion of MUSs to indeterminate strings by first simplifying the notion of an occurrence in indeterminate strings from ED strings. For that, we say that a string $P \in \Sigma^*$ has an *occurrence* in an indeterminate string $\widetilde{S}$ at position i if there exists a string $X \in \mathcal{L}(\widetilde{S})$ with $X[i..i+|P|-1] = P$. We say a string P in $\widetilde{S}$ is *unique* if it has only one occurrence, meaning that the position of its occurrence is uniquely determined. Further, we call P a *minimal unique substring (MUS)* if every proper substring X of P has at least two occurrences in $\widetilde{S}$.

We show that the decision version of finding a unique substring of a specific length in an indeterminate string is NP-hard. The optimization version of minimizing the length then gives a MUS.

▶ **Problem 1** (Unique Substring problem). *Given an indeterminate string $\widetilde{S}$ and an integer k, the Unique Substring problem is to decide whether there is a unique substring of $\widetilde{S}$ with length at most k.*

▶ **Theorem 1.** *Unique Substring is NP-hard for $\sigma \geq 3$ and $r \geq 2$.*

Proof. Given a 3-CNF $F = C_1 \wedge \cdots \wedge C_m$ with n variables and m clauses. We construct an indeterminate string $\widetilde{S}$ over the alphabet $\{0, 1, \$\}$ such that there is a unique substring of $\widetilde{S}$ of length at most n if and only if F is satisfiable. We assume the variables are $x_1, \ldots, x_n$.

Let ℓ_a, ℓ_b, and ℓ_c be the literals of the clause $C_j = (\ell_a \vee \ell_b \vee \ell_c)$ such that ℓ_i is the literal of the variable x_i for $i \in \{a, b, c\}$. For each i, we set $v_i = 0$ if ℓ_i is positive ($\ell_i = x_i$) and $v_i = 1$ otherwise ($\ell_i = \neg x_i$), and construct the following indeterminate string $\widetilde{T_j}$ based on the v_i's.

$$\widetilde{T_j} = \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{a-1} \{ v_a \} \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{b-a-1} \{ v_b \} \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{c-b-1} \{ v_c \} \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{n-c}.$$

Then, $\widetilde{S}$ is given as follows:

$$\widetilde{S} = \widetilde{T_1} \{ \$ \} \widetilde{T_2} \{ \$ \} \ldots \{ \$ \} \widetilde{T_{m-1}} \{ \$ \} \widetilde{T_m} \{ \$ \} \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{n} \{ \$ \} \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{n-1} \{ \$ \} \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{n-1}.$$

$\widetilde{S}$ has $\mathcal{O}(nm)$ symbols because each $\widetilde{T_j}$ has n symbols. We make the following observations on $\widetilde{S}$:

- Each bitstring, i.e., a string of the alphabet $\{0, 1\}$, of length n occurs at least once, namely in the symbol $\left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^{n}$ at $\widetilde{S}[1 + (n+1)m]$.

- Each bitstring shorter than n occurs more than once.
- No substring of length at most n contains more than one "\$".
- Every string of length at most n containing a "\$" occurs more than once.
- A bitstring $B[1..n]$ of length n encodes an assignment with $B[i] = 1$ if and only if x_i is true. Such a bitstring B occurs more than once in $\widetilde{S}$ if and only if its encoded assignment falsifies F. Specifically, B occurs in $\widetilde{T_i}$ if its assignment falsifies C_i.

Therefore, there can be no unique substring of length less than n, and a unique substring of length n exists if and only if F is satisfiable. ◀

▶ **Example 2.** Given the 3-CNF $F = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$ with $n = 4$ variables x_1, x_2, x_3, x_4. The indeterminate string $\widetilde{S}$ for F as described in the proof of Thm. 1 is

$$\{0\}\{1\}\{0\}\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}\{\$\}\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}\{0\}\{1\}\{0\}\{\$\}\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}^4\{\$\}\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}^3\{\$\}\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}^3.$$

The string 1000 ($x_1 = 1, x_2 = x_3 = x_4 = 0$) is a MUS because it occurs only once in $\widetilde{S}$, all strings of length 3 on the alphabet $\{0, 1\}$ occur in $\widetilde{S}$. Other strings of length 4 such as 0100 and 00\$1 occur at least twice. Thus, 1000 is one of the shortest MUS in $\widetilde{S}$.

4.2 Hardness of Minimal Absent Word

An *absent word* of a classic string T is a non-empty string that is not a substring of T. An absent word X of T is called a *minimal absent word* (*MAW*) of T if all proper substrings of X occur in T. More general, we say that a string in an indeterminate string $\widetilde{S}$ is *absent* if it does not occur in $\widetilde{S}$. An absent string X is *minimal* in $\widetilde{S}$ if every proper substring of X occurs in $\widetilde{S}$. In what follows, we show that the decision problem for finding an absent word of at most a specific length is NP-hard.

▶ **Problem 2** (Absent Word problem). *Given an indeterminate string $\widetilde{S}$ and an integer k, the k-Absent Word problem is to decide whether there is an absent word of $\widetilde{S}$ with length at most k.*

▶ **Theorem 3.** *Absent Word is NP-hard for $\sigma \geq 3$ and $r \geq 3$.*

Proof. Our idea is to follow the proof of Theorem 1, except that we define $\widetilde{S}$ as

$$\widetilde{S} = \widetilde{T_1}\{\$\}\widetilde{T_2}\{\$\}\ldots\{\$\}\widetilde{T_{m-1}}\{\$\}\widetilde{T_m}\{\$\}\left\{\begin{smallmatrix}0\\1\\\$\end{smallmatrix}\right\}^{n-1}\{\$\}\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}^{n-1}.$$

$\widetilde{S}$ contains all strings from the alphabet $\{0, 1, \$\}$ with a length of at most $n-1$ as a substring. An absent word thus must be of length at least n. Since any string of length at most n containing a \$ occurs also in the last $2n + 1$ symbols of $\widetilde{S}$, i.e., $\left\{\begin{smallmatrix}0\\1\\\$\end{smallmatrix}\right\}^{n-1}\{\$\}\left\{\begin{smallmatrix}0\\1\end{smallmatrix}\right\}^{n-1}$, the search for an absent word of length at most n boils down to finding a bitstring of length n. Each bitstring B encodes an assignment. If the assignment does not satisfy clause C_i, then B matches $\widetilde{T_i}$. On the one hand, if there is no satisfying assignment for the CNF, then all substrings of length n with alphabet $\{0, 1\}$ must be present in $\widetilde{S}$. In total, there is no absent word of length at most n in $\widetilde{S}$. On the other hand, if there is a satisfying assignment, there is also an absent word of length n, which is the bitstring encoding this assignment. ◀

▶ **Example 4.** We reuse the 3-CNF F from Example 2, i.e., $F = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$. The indeterminate string $\widetilde{S}$ for F as described above is

$$\widetilde{S} = \{0\}\{1\}\{0\}\left\{\begin{matrix}0\\1\end{matrix}\right\}\{\$\}\left\{\begin{matrix}0\\1\end{matrix}\right\}\{0\}\{1\}\{0\}\{\$\}\left\{\begin{matrix}0\\1\\\$\end{matrix}\right\}^3\{\$\}\left\{\begin{matrix}0\\1\end{matrix}\right\}^3.$$

The string 1000 ($x_1 = 1, x_2 = x_3 = x_4 = 0$) does not occur in $\widetilde{S}$. However, its longest proper prefix 100 and suffix 000 occur each in $\widetilde{S}$. Hence, 1000 is a MAW. It is also a shortest MAW since each string of length 3 appears in $\widetilde{S}$.

The Absent Word problem is FPT in the size s of the language $\mathcal{L}(\widetilde{S})$. The algorithm of [88] computes a generalized MAW of a set of strings $\mathcal{S}$ in $\mathcal{O}(m)$ time, where m is the total length of all strings in $\mathcal{S}$. Here, a *generalized MAW* is a string that is a MAW for all strings in $\mathcal{S}$. Hence, computing the generalized MAW of $\mathcal{L}(\widetilde{S})$ takes $\mathcal{O}(ns)$ time, and the computed generalized MAW is a MAW of $\widetilde{S}$.

5 Hardness of k-Anti-Power

A factorization of a classic string T is a partition of T into factors $F_1, \ldots, F_k$ such that their concatenation gives T, i.e., $F_1 \cdots F_k = T$. T is an *anti-power* of order k if T can be factorized into k factors of equal length such that all factors are distinct. As a generalization, we say that a GD string has a *k-anti-power* if its language contains a k-anti-power.

▶ **Problem 3** (Anti-Power problem)**.** *Given a GD string $\widetilde{S}$ and an integer k, the Anti-Power problem is to decide whether $\widetilde{S}$ has an anti-power of order k.*

If the cardinality of $\mathcal{L}(\widetilde{S})$ is polynomial in the length $m(= \sum_{i=1}^{n} |\widetilde{S}[i][1]|)$ of a string in $\mathcal{L}(\widetilde{S})$, then we check each string in $\mathcal{L}(\widetilde{S})$ one by one for being a k-anti-power. The time per check is $\mathcal{O}(m)$, and thus we obtain polynomial time in m times the cardinality of $\mathcal{L}(\widetilde{S})$. However, the cardinality of $\mathcal{L}(\widetilde{S})$ can be exponential.

▶ **Theorem 5.** *Anti-Power is NP-hard for $\sigma \geq 2$ and $r \geq n$.*

Proof. We reduce from Hamiltonian Path, which is one of Karp's classic NP-hard problems [72]. Given an undirected graph $G = (V, E)$ with $n = |V| \geq 2$ without self-loops, we construct a GD string $\widetilde{S}$ such that $\widetilde{S}$ has an anti-power of order $k = 2(\binom{n}{2} - |E| + n) - 1$ if and only if G has a Hamiltonian path. For this, we assume that we can enumerate the vertices such that $V = \{v_1, \ldots, v_n\}$. Then we can identify each vertex $v_i \in V$ with a character c_i of an alphabet $\Sigma = \{c_1, \ldots, c_n\}$. Let $\overline{E} = \binom{V}{2} \setminus E = \{e_1, \ldots, e_{|\overline{E}|}\}$ be the complement of E, again without self-loops. For each $e_i = \{u_i, v_i\} \in \overline{E}$ we define the string $S_i = c_{u_i}c_{v_i} \cdot c_{v_i}c_{u_i} \in \Sigma^4$. Then define the GD string $\widetilde{S}$ as

$$\widetilde{S} = S_1 \cdots S_{|\overline{E}|}\left\{\begin{matrix}c_1^3\\\vdots\\c_n^3\end{matrix}\right\}\left\{\begin{matrix}c_1^4\\\vdots\\c_n^4\end{matrix}\right\}^{n-2}\left\{\begin{matrix}c_1^3\\\vdots\\c_n^3\end{matrix}\right\}.$$

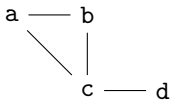
We make the following observations on $\widetilde{S}$.

- The length of each string of the language $\mathcal{L}(\widetilde{S})$ is $4(\binom{n}{2} - |E| + n) - 2$ due to the following two reasons, where we split the analysis by the prefix of the S_i's and the remaining suffix. First, the cardinality of $\overline{E}$ is $\binom{n}{2} - |E|$ and therefore the summation of the lengths of all S_i's accumulates to $4|\overline{E}| = 4(\binom{n}{2} - |E|)$. The remaining suffix has length $3 + 4(n-2) + 3 = 4n - 2$. The summation of both is $4(\binom{n}{2} - |E| + n) - 2$, which is $2k$.

- A k-anti-power X of $\widetilde{S}$ is a member of $\mathcal{L}(\widetilde{S})$, and thus must have length $2k$. In particular, X consists of k factors of equal length 2. By construction, each factor of X starts at an odd position in $\widetilde{S}$.
- Every string $Y \in \mathcal{L}(\widetilde{S})$ corresponds to a walk $W[1..n]$ over n nodes in the complete graph $\mathcal{G}_{\mathrm{C}} := (V, \binom{V}{2})$ without self-loops, and vice versa: To understand that, let us focus on the suffix $\widetilde{S}[4|\overline{E}|+1..]$ directly after the sequence of S_i's. For each $j \in [1..n]$, let $s_j = 1 + 4|\overline{E}| + 4(j-1)$. Then $Y[s_j..s_j+1]$ is of the form $c_a c_a$ for some $a \in [1..n]$ and encodes that the j-th visited node of W is $W[j] = v_a$. By construction for $j < n$, $Y[s_j+2..s_j+3]$ is of the form $c_a c_b$ where $W[j] = v_a$ and $W[j+1] = v_b$ are the j-th and $(j+1)$-st node on W, respectively. Now take a walk $W[1..n]$ over n nodes in $\mathcal{G}_{\mathrm{C}}$ represented by a string Y. W is a permutation if and only if it is Hamiltonian in $\mathcal{G}_{\mathrm{C}}$ by the pigeonhole principle – otherwise a node has to appear twice in $W[1..n]$
- If and only if W is not Hamiltonian in $\mathcal{G}_{\mathrm{C}}$, W visits at least one node i more than once. In this case, $c_i c_i$ occurs twice in Y, starting at odd positions. Therefore, Y cannot be a k-anti-power.
- If and only if W uses an edge from v_a to v_b with $a \neq b$, there is some j such that $Y[s_j..s_j+5] = c_a c_a c_a c_b c_b c_b$ for the string $Y \in \mathcal{L}(\widetilde{S})$ corresponding to W. Assume that this edge $\{v_a, v_b\}$ is not in E. Since $\{v_a, v_b\} \in \overline{E}$, $c_a c_b$ also occurs starting at some odd position $p < 4|\overline{E}|$ in Y, more precisely in the part of Y generated by an S_i. Therefore, Y cannot be a k-anti-power.
- Because of the previous two statements, a Hamiltonian path in (V, E) corresponds to a k-anti-power. Since every walk of length n (and thus every Hamiltonian path) corresponds to a string in $\mathcal{L}(\widetilde{S})$, the converse is also true.

Finally, we can generalize the proof to GD strings over the binary alphabet $\{0,1\}$ by replacing each c_i with a unique bitstring ($\in \{0,1\}^*$) of length $\lceil \log_2 n \rceil$. The anti-power we aim to compute is still of order k, but each factor then contains $\lceil \log_2 n \rceil$ bits. ◀

▶ **Example 6.** Consider the following graph, with characters $V = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ instead of numbers as node labels for readability.



The set of edges is $E = \{\{\mathtt{a},\mathtt{b}\}, \{\mathtt{a},\mathtt{c}\}, \{\mathtt{b},\mathtt{c}\}, \{\mathtt{c},\mathtt{d}\}\}$. Its complement, omitting self-loops, is $\overline{E} = \{\{\mathtt{a},\mathtt{d}\}, \{\mathtt{b},\mathtt{d}\}\}$. The corresponding GD string is

$$\widetilde{S} = \{\,\mathtt{ad}\,\}\{\,\mathtt{da}\,\}\{\,\mathtt{bd}\,\}\{\,\mathtt{db}\,\}\left\{\begin{matrix}\mathtt{aaa}\\ \mathtt{bbb}\\ \mathtt{ccc}\\ \mathtt{ddd}\end{matrix}\right\}\left\{\begin{matrix}\mathtt{aaaa}\\ \mathtt{bbbb}\\ \mathtt{cccc}\\ \mathtt{dddd}\end{matrix}\right\}\left\{\begin{matrix}\mathtt{aaaa}\\ \mathtt{bbbb}\\ \mathtt{cccc}\\ \mathtt{dddd}\end{matrix}\right\}\left\{\begin{matrix}\mathtt{aaa}\\ \mathtt{bbb}\\ \mathtt{ccc}\\ \mathtt{ddd}\end{matrix}\right\}.$$

We color the characters of $\widetilde{S}$ to make the example more readable. Characters in yellow color (■) encode the visited nodes of a walk, and characters in blue color (■) encode the edges between the visited nodes. There are four anti-powers of $\widetilde{S}$ of order $k = 11$, namely

- addabddbaaabbbbccccddd for $\mathtt{a} \to \mathtt{b} \to \mathtt{c} \to \mathtt{d}$,
- addabddbbbbaaaaccccddd for $\mathtt{b} \to \mathtt{a} \to \mathtt{c} \to \mathtt{d}$,
- addabddbdddccccaaaabbb for $\mathtt{d} \to \mathtt{c} \to \mathtt{a} \to \mathtt{b}$, and
- addabddbdddccccbbbbaaa for $\mathtt{d} \to \mathtt{c} \to \mathtt{b} \to \mathtt{a}$.

From each anti-power we can construct a Hamiltonian path by reading each same-character pair in yellow color (■).

6 Hardness of Longest Previous Factor

The LPF problem on a classic string $T[1..n]$ is to compute, for a given position $T[i]$, the length of a substring starting before i that shares the longest common prefix with $T[i..]$, i.e.,

$$\mathsf{LPF}[i] := \max\{\ell \in [0..n-i+1] \mid T[j..j+\ell-1] = T[i..i+\ell-1] \wedge j < i\}.$$

$\mathsf{LPF}[i]$ is the length of the *longest previous factor (LPF)* of $T[i]$. We can translate this problem to ED strings by a weak and a strong variant. Given a text-position (i,j,k) in an ED string $\widetilde{S}$, the weak variant states that an LPF is the longest string that has an occurrence at (i,j,k) and an earlier occurrence at $(i',j',k') < (i,j,k)$, where $(i',j',k') < (i,j,k)$ if $i' < i$ or ($i' = i$, $j' = j$, and $k' < k$). Informally, the strong variant requires the LPF to choose the same path through $\widetilde{S}$ in case it takes characters from $\widetilde{S}[i]$. More technically, the LPF of (i,j,k) is the LPF at position x of a classic string X in the language $\mathcal{L}(\widetilde{S})$ such that X covers (i,j,k) at position x. With covering we mean that there is a factorization $X = X_1 \cdots X_n$ such that X_z is a (possibly empty) substring of X with $X_z \in S[z]$ for each $z \in [1..n]$ and $X_i = \widetilde{S}[i][j]$ with $|X_1 \cdots X_{i-1}| + k = x$. Consider for example $\widetilde{S} = \{\mathtt{ab}\}\left\{\begin{smallmatrix}\mathtt{abc}\\ \mathtt{c}\end{smallmatrix}\right\}$. In the strong variant, the LPF of text-position $(2,1,1)$ is $\mathtt{ab}$ because there is only one string $X = \mathtt{ababc} \in \mathcal{L}(\widetilde{S})$ covering $(2,1,1)$ at position 3, and the LPF of $X[3]$ is $\mathtt{ab}$. In the weak variant, $\mathtt{abc}$ is also permissible. The weak variant can be reduced to computing LCEs, and therefore is solvable in quadratic time (cf. Appendix A in the appendix). This strong variant is at least as hard as computing the maximum length ℓ of the LPFs of the strings in $\mathcal{L}(\widetilde{S})$, i.e., $\ell = \max\{\max\{\mathsf{LPF}(T)[i] \mid 1 \le i \le |T|\} \mid T \in \mathcal{L}(\widetilde{S})\}$. While we can compute ℓ for indeterminate strings in polynomial time (see Appendix B), we show that computing ℓ is NP-hard, formalized by the following decision problem:

▶ **Problem 4** (LPF problem). *Given an ED string $\widetilde{S}$ and an integer k, the LPF problem is to decide whether the longest LPF in $\mathcal{L}(\widetilde{S})$ is of length at least k.*

Like Problem 3, LPF can be answered in polynomial time if $\mathcal{L}(\widetilde{S})$ has polynomial cardinality because we can compute the LPF array for each string in $\mathcal{L}(\widetilde{S})$ in linear time to its length [39].

▶ **Theorem 7.** *LPF is NP-hard for $\sigma \ge 3$ and $r \ge \sigma + 1$.*

Proof. We first define two concepts: periodicity and roots. A string S is called *periodic* if the largest possible rational number $k \ge 1$ with $S = R^k$ for a string R is at least two; in that case we call R the *root* of S.

For the proof, we reduce from Common Subsequence. Let $\mathcal{S}$ be a set of strings and $k \ge 1$ an integer. Common Subsequence asks whether there is a string of length k that is a subsequence common to all strings in $\mathcal{S}$. Common Subsequence is NP-hard [78] in the cardinality of $\mathcal{S}$ for alphabet sizes of at least 2.

The general idea is to construct an ED string such that every sufficiently long repeated factor F overlaps and therefore is periodic, such that F's root is a subsequence of length k (modulo a separator symbol) of some or all strings in $\mathcal{S}$. In what follows, we show that the longest such factor F is a subsequence of length k common to *all* strings in $\mathcal{S}$.

Assume that all strings in $\mathcal{S} = \{S_1, \ldots, S_f\}$ are over the alphabet Σ. We now construct an ED string $\widetilde{S}$ that has a longest LPF of length $(k+1)(f+\ell)+1$ if and only if $\mathcal{S}$ has a common subsequence of length k. For each $i \in [1..f]$ we define $\widetilde{S_i} = \left\{\begin{smallmatrix}S_i[1]\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}S_i[2]\\ \varepsilon\end{smallmatrix}\right\}\cdots\left\{\begin{smallmatrix}S_i[|S_i|]\\ \varepsilon\end{smallmatrix}\right\}$. By construction, $\mathcal{L}(\widetilde{S_i})$ is the set of subsequences of S_i. Let \$ be a symbol not in Σ. Let

$\ell \geq 3$ be minimal such that $k \cdot (\ell - 2) > \sum_i |S_i| = ||\mathcal{S}||$, where $||\mathcal{S}||$ denotes the accumulated lengths of all strings in $\mathcal{S}$. By misusing notation, we interpret Σ as an ED symbol matching any character of Σ, and denote the ED symbol matching any character of Σ and the empty word by $\bar{\Sigma} = \Sigma \cup \{\varepsilon\}$. Then define

$$\widetilde{S} = \left(\left\{\,\$\,\right\}\bar{\Sigma}^k\right)^\ell \left\{\,\$\,\right\}\widetilde{S_1}\left\{\,\$\,\right\}\cdots\left\{\,\$\,\right\}\widetilde{S_f}\left\{\,\$\,\right\}\Sigma^k\left\{\,\$\,\right\}.$$

By the minimality of the chosen value of ℓ, $||\widetilde{S}|| \in \mathcal{O}(\sigma||\mathcal{S}||)$. Now consider the LPF of the position of the second occurrence of $\left\{\,\$\,\right\}$ in $\widetilde{S}$, i.e., symbol $\widetilde{S}[k+2]$.

- The only non-empty previous factor F of $\widetilde{S}[k+2]$ starts at the very beginning of $\widetilde{S}$ because it must start with "\$". Assume that F is the longest possible such factor.
- On the one hand, F has length at least $(\ell-1)(k+1)+1$ due to the prefix $\left(\left\{\,\$\,\right\}\Sigma^k\right)^\ell$ of $\widetilde{S}$ with $\ell \geq 3$ matching the first $(\ell-1)(k+1)+1$ symbols starting at $\widetilde{S}[k+2]$.
- On the other hand, F is of the form $(\$W)^z\$(W[1..p])$ for some $z \geq 0$, $p \in [0..k-1]$, $W \in \Sigma^k$ because the occurrences of F starting at $\widetilde{S}[1]$ and at the second $\left\{\,\$\,\right\}$ overlap.

If $|F| = (k+1)(f+\ell)+1$, W is a common subsequence of all strings in $\mathcal{S}$. That is because F is a prefix of a string $X \in \mathcal{L}(\widetilde{S})$ by the definition of the strong variant of LPF, and therefore its substrings Y_i and Y_{i+1} matching with $\widetilde{S_i}$ and $\widetilde{S_{i+1}}$, respectively, must match, i.e., $Y_i = Y_{i+1}$ for every $i \in [1..f-1]$. Thus, $W = Y_i$ is determined by $\widetilde{S_i}$, for all i. Finally, F cannot be longer than $(k+1)(f+\ell)+1$ due to the number of \$'s and since every $(k+1)$-st character of F is \$. It is left to show that all other text-positions have a shorter LPF.

- An occurrence of a string starting after the prefix $\left(\left\{\,\$\,\right\}\Sigma^k\right)^{\ell-1}$ of $\widetilde{S}$ can have length at most

$$\begin{aligned} 3 + 2k + \sum_{i=1}^{f}(1+|S_i|) &= 3 + 2k + f + \sum_{i=1}^{f}|S_i| \\ &< 3 + 2k + f + k(\ell-2) = \ 3 + f + k\ell \\ &< kf + f + k\ell + \ell + 1 = \ (k+1)(f+\ell)+1 \end{aligned}$$

- An occurrence of a string starting at a text-position p before $\left(\left\{\,\$\,\right\}\Sigma^k\right)^{\ell-1}$ with length of at least $2k+2$ has a substring matching $\left\{\,\$\,\right\}\Sigma^k\left\{\,\$\,\right\}$. A sufficiently long LPF for text-position p must overlap by construction, and thus again every $(k+1)$-st character must be \$. When p is after the second \$, the LPF of P is shorter than $(k+1)(f+\ell)+1$. ◀

▶ **Example 8.** As an example for the proof of Theorem 7 consider $\mathcal{S} = \{S_1, S_2\}$ with $f = |\mathcal{S}| = 2$, $S_1 = \texttt{abb}$ and $S_2 = \texttt{bab}$. We want to test whether $\mathcal{S}$ has a common subsequence of length $k = 2$. Therefore, we need to select the smallest $\ell > (|S_1| + |S_2|)/k + 2 = 5$, which is $\ell = 6$. Then $\widetilde{S_1} = \left\{\begin{smallmatrix}\texttt{a}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}$, $\widetilde{S_2} = \left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{a}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}$, and

$$\widetilde{S} = \left(\$\left\{\begin{smallmatrix}\texttt{a}\\ \texttt{b}\\ \varepsilon\end{smallmatrix}\right\}^2\right)^6 \$\widetilde{S_1}\$\widetilde{S_2}\$\left\{\begin{smallmatrix}\texttt{a}\\ \texttt{b}\end{smallmatrix}\right\}^2\$$$

$$= \$\left\{\begin{smallmatrix}\texttt{a}\\ \texttt{b}\\ \varepsilon\end{smallmatrix}\right\}^2\$\left(\left\{\begin{smallmatrix}\texttt{a}\\ \texttt{b}\\ \varepsilon\end{smallmatrix}\right\}^2\$\right)^5 \left\{\begin{smallmatrix}\texttt{a}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}\$\left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{a}\\ \varepsilon\end{smallmatrix}\right\}\left\{\begin{smallmatrix}\texttt{b}\\ \varepsilon\end{smallmatrix}\right\}\$\left\{\begin{smallmatrix}\texttt{a}\\ \texttt{b}\end{smallmatrix}\right\}^2\$.$$

The second occurrence of \$, i.e., $\widetilde{S}[4] = \$$ has the LPF $(\texttt{\$ab})^8\$$ starting at $S[1]$. This length is $(k+1)(f+\ell)+1$, and thus $\texttt{ab}$ must be a common subsequence of S_1 and S_2 of length $k = 2$.

7 LZ77

A natural extension of the LZ77 factorization from strings to ED strings is to define the LZ factorization on ED strings as a set of paths that cover the characters of the ED string. More precisely, given an ED string $\widetilde{S}$, we define a graph $G = (V, E)$ such that each character $\widetilde{S}[i][j][k]$ is a vertex $v_{i,j,k} \in V$. There is a directed edge from $v_{i,j,k}$ to $v_{i',j',k'}$ if and only if

- $i = i'$ and $j = j'$ and $k + 1 = k'$, or
- $i + 1 = i'$ and $k' = 1$ and $|\widetilde{S}[i][j]| = k$.

A path factorization of $\widetilde{S}$ is then defined as a set of paths in G such that each vertex in V is covered by at least one path. We call a path factorization an *LZ factorization* if each path of the factorization starting at position $v_{i,j,k}$ has a previous occurrence ending before $\widetilde{S}[i]$, i.e., there exists a path ending at some position $v_{i',j',k'}$ with $i' < i$ such that the sequence of characters on both paths are identical, and this occurrence is the longest possible.

▶ **Problem 5** (LZ problem). *Given an indeterminate string $\widetilde{S}$ and an integer k, the LZ problem is to decide whether $\widetilde{S}$ has an LZ factorization with k paths.*

We here focus on two flavors of the LZ problem, which differ in whether we are allowed to cover the same character positions multiple times by different factor paths or not. In other words, the restricted version enforces a partition of the ED string by factor paths.

7.1 Restricted Version

For the restricted version, we obtain hardness by reducing from the 3-dimensional matching problem (3DM) [56, Sect. 3.1.2]. In 3DM we are given three finite sets X, Y, and Z of the same size k. Let T be a subset of $X \times Y \times Z$. This means that T consists of triples (x, y, z) where $x \in X$, $y \in Y$, and $z \in Z$. It is known that finding a perfect matching in T is NP-complete [72]. Here, a subset $M \subseteq T$ is called a *matching* if, for any two distinct triples $(x_1, y_1, z_1) \in M$ and $(x_2, y_2, z_2) \in M$, it holds that $x_1 \neq x_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$. M is further called a *perfect matching* if every element of X, Y, and Z appears in exactly one triple in M.

For the reduction, we assume that X, Y, and Z all have different elements, and treat the three sets as disjoint alphabets. Next, we construct a string P as follows, which is initially empty. For each triplet $(x_i, y_i, z_i) \in T$, append $x_i y_i z_i$ to P. Finally, append a symbol \$ to P that nowhere else appears in P, and define $\widetilde{S} := PXYZ$, where we interpret X, Y, and Z each as indeterminate symbols. Now, the LZ77 factorization without overlaps has $z_P + k$ factors if and only if T has a perfect matching, where z_P is the number of LZ77 factors of $P\$$.

▶ **Theorem 9.** *The restricted LZ problem is NP-hard for indeterminate strings with an alphabet size of $\Theta(n)$ and $r = \Theta(n)$.*

▶ **Example 10.** Given three sets $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$, and $Z = \{z_1, z_2\}$, and the triplet set $T = \{(x_1, y_1, z_2), (x_1, y_2, z_1), (x_2, y_2, z_1)\}$, we construct the string $P = \texttt{x}_1\texttt{y}_1\texttt{z}_2\texttt{x}_1\texttt{y}_2\texttt{z}_1\texttt{x}_2\texttt{y}_2\texttt{z}_1\$$. Then, we have $\widetilde{S} = PXYZ$. The LZ77 factorization of $P\$$ has $z_P = 9$ factors as follows: $\texttt{x}_1 \mid \texttt{y}_1 \mid \texttt{z}_2 \mid \texttt{x}_1 \mid \texttt{y}_2 \mid \texttt{z}_1 \mid \texttt{x}_2 \mid \texttt{y}_2 \cdot \texttt{z}_1 \mid \$$, where "|" separates the factors and '·' highlights concatenation. Since T has a perfect matching $M = \{(x_1, y_1, z_2), (x_2, y_2, z_1)\}$, we can cover the remaining part XYZ of $\widetilde{S}$ with $k = 2$ additional LZ paths: $\texttt{x}_1 \cdot \texttt{y}_1 \cdot \texttt{z}_2$ and $\texttt{x}_2 \cdot \texttt{y}_2 \cdot \texttt{z}_1$. Thus, the total number of LZ paths to cover $\widetilde{S}$ is $z_P + k = 11$.

7.2 Unrestricted Version

In case that we allow paths to overlap the same node, the above reduction does not work anymore. Instead, we reduce from set cover for this problem. Set cover asks, given n elements and m sets and an integer k, whether there exist k sets (among the m ones) whose union contains all n elements.

Suppose that the elements are enumerated $x_1, \dots, x_n$, and let y be a new element. Define the list $L_j[1..n]$ with $L_j[i] = x_i$ if x_i is in the j-th set, and otherwise $L_j[i] = y$. Further, define

$$\widetilde{S} = \left\{ \begin{array}{c} L_1 \\ L_2 \\ \vdots \\ L_n \\ y^{n+1} \end{array} \right\} \left\{ \begin{array}{c} x_1 \\ y \end{array} \right\} \left\{ \begin{array}{c} x_2 \\ y \end{array} \right\} \cdots \left\{ \begin{array}{c} x_n \\ y \end{array} \right\} \{ y \}.$$

We now analyze the number of LZ paths to cover $\widetilde{S}$. Since $\widetilde{S}[1]$ is the first ED symbol, it does not contain any LZ paths by definition. Next we focus on the path y^{n+1} in $\widetilde{S}[2..]$. It is possible to cover these characters with one LZ path using the last string in $\widetilde{S}[1]$ as a reference. We actually have to select this path or any suffix of it to cover the last character y in $\widetilde{S}$. Finally, to cover all characters in $\widetilde{S}[2..n]$ except those in y, we need to select a reference from $\widetilde{S}[1]$ that contains the respective character x_i to cover $\widetilde{S}[i+1]$. However, each such reference selections corresponds to the selection of a set in the set cover problem, by ignoring the y characters, which have already been covered by the path for y^{n+1}. Consequently, the number of paths needed for covering the $\widetilde{S}[2..]$ is exactly the size of the minimum set cover plus one.

▶ **Theorem 11.** *The unrestricted* LZ *problem is NP-hard for GD strings with an alphabet size of* $\Theta(n)$ *and* $r = \Theta(n)$.

▶ **Example 12.** Consider the set cover instance with elements $\{a, b, c, d\}$ and sets $\{a, b\}$, $\{b, c\}$, $\{c, d\}$, and $\{a, d\}$. The minimum set cover has size 2, e.g., by selecting the sets $\{a, b\}$ and $\{c, d\}$. We construct the GD string $\widetilde{S}$ below.

To cover $\widetilde{S}$ with LZ paths, we first select the path $yyyyy$ from the last string in $\widetilde{S}[1]$ to cover the last character y in $\widetilde{S}$. Then, to cover the other characters, we need to select two more paths from $\widetilde{S}[1]$, e.g., $\widetilde{S}[1][1] = abyy$ and $\widetilde{S}[1][3] = cd$, which correspond to the selected sets in the set cover instance. Thus, we need at least three LZ paths to cover $\widetilde{S}$.

$$\widetilde{S} = \left\{ \begin{array}{c} abyy \\ ybcy \\ yycd \\ ayyd \\ yyyyy \end{array} \right\} \left\{ \begin{array}{c} a \\ y \end{array} \right\} \left\{ \begin{array}{c} b \\ y \end{array} \right\} \left\{ \begin{array}{c} c \\ y \end{array} \right\} \left\{ \begin{array}{c} d \\ y \end{array} \right\} \{ y \}.$$

8 SAT Formulation

In what follows, we present SAT formulations for computing a unique substring or an absent word of a specified length x in an indeterminate string. The SAT formulations can be transformed into a MAX-SAT formulation by an optimizing objective on $x \in [1..n]$. For the formulations, we represent a symbol of an indeterminate string $\widetilde{T}[1..n]$ as a list of characters such that $\widetilde{T}[i]$ is a list. We let $\widetilde{T}[i][k]$ denote the k-th character in the list $\widetilde{T}[i]$ and $|\widetilde{T}[i]|$ the

number of characters stored in $\widetilde{T}[i]$. We interpret the Boolean variables true and false as integers 1 and 0 to allow us to use expressions such as summations. We model our answer as a string $X[1..x] \in \Sigma^x$ of length x. The length x can be given (decision problem) or is object to optimization to obtain the shortest length. We can model x by n Boolean variables x_i as follows.

$$\sum_{i=1}^{n} x_i = 1. \qquad \text{(LENX)} \qquad \min\{i \in [1..n] : x_i = 1\}. \qquad \text{(MINX)}$$

In the following, variables that are not defined are assumed to be false (not set).

8.1 Shortest Minimal Absent Word (MAW)

X can be expressed by $x\sigma$ Boolean variables arranged in a matrix $X'[1..x][1..\sigma]$ choosing characters from $\Sigma = \{1, \ldots, \sigma\}$.

$$\forall \ell \in [1..x] : \sum_{c=1}^{\sigma} X'[\ell, c] = 1. \qquad \text{(SETX)}$$

$$\{\mathcal{O}(x),\ \mathcal{O}(\sigma)\}$$

The gray curly brackets denote two asymptotic upper bounds, the first is on the number of generated clauses and the second on the maximum size of each such clause. Here, we generate for each X-position a clause of size σ.

To check whether X is absent, we need to verify that there is no occurrence of X starting at any text position $t \in [1..n - x + 1]$ of $\widetilde{T}$. In other words, we check for each such t that there is an X-position $\ell \in [1..x]$ such that $X[\ell] \notin \widetilde{T}[t + \ell - 1]$. To express that as a formula, we create a three-dimensional grid of boolean variables $M[k, t, \ell]$ modelling these mismatches. We set $M[k, t, \ell]$ to true if $X[\ell] \neq \widetilde{T}[t + \ell - 1][k]$, where $\widetilde{T}[t + \ell - 1][k]$ denotes the k-th character in the list $\widetilde{T}[t + \ell - 1]$, if it exists.

$$\forall \ell \in [1..x], t \in [1..n - x + 1], k \in [1..|\widetilde{T}[t + \ell - 1]|] : X[\ell] \neq \widetilde{T}[t + \ell - 1][k] \implies M[k, t, \ell]. \qquad \text{(M)}$$

$$\{\mathcal{O}(xnr),\ \mathcal{O}(1)\}$$

Recall that $r = r(\widetilde{T}) \leq \sigma$ is the maximum number of characters a list $\widetilde{T}[i]$ can store. Next we reduce $M[k, t, \ell]$ to $M'[t, \ell]$ by setting $M'[t, \ell]$ to true only if $X[\ell]$ mismatches with all characters in the list $\widetilde{T}[t + \ell - 1]$, i.e.,

$$\forall \ell \in [1..x], t \in [1..n - x + 1] : \sum_{k=1}^{|\widetilde{T}[t+\ell-1]|} M[k, t, \ell] = |\widetilde{T}[t + \ell - 1]| \implies M'[t, \ell]. \qquad \text{(M')}$$

$$\{\mathcal{O}(xn),\ \mathcal{O}(r)\}$$

So $M'[t, \ell]$ is true if and only if $X[\ell] \notin \widetilde{T}[t + \ell - 1]$. Finally, we require that there is some X-position ℓ for which we cannot match $X[\ell]$ with any of the characters of $\widetilde{T}[t + \ell - 1]$, i.e.,

$$\forall t \in [1..n - x + 1] : \sum_{\ell=1}^{x} M'[t, \ell] \neq 0. \qquad \text{(CONS)}$$

$$\{\mathcal{O}(n),\ \mathcal{O}(x)\}$$

Table 1 Computation of a shortest MAW and MUS on a randomly generated indeterminate string $\widetilde{S}$ with $\sigma = 4$ and $r(\widetilde{S}) = 2$. Time is in seconds, and n is the length of the prefix $\widetilde{T}$ of $\widetilde{S}$ taken as input for computation.

	MAW		MUS	
n	time	x	time	x
10	0.02	2	0.03	2
100	169.62	4	171.60	3
150	566.88	4	580.68	4
200	1365.87	4	1376.11	4
250	2615.42	5	2708.00	5

If Eq. (CONS) holds, then X must be an absent word. In total, we have $x\sigma + n$ selectable Boolean variables. The number of clauses is in $\mathcal{O}(xnr)$. The same upper bound has the size of the CNF being the summation of the sizes of all clauses.

Listing 1 shows an encoding in answer set programming (ASP), where each line is commented by one of the above equations.

Listing 1 ASP encoding computing a MAW. The last line (command `show`) is for the output.

```
1 { len(X) : X = 1..n } 1. %(LENX)
#minimize { X : len(X) }. %(MINX)
1 { x(L,C) : a(C) } 1 :- len(X), L=1..X. %(SETX)
m(K,T,L) :- t(K,T+L-1,C), x(L,D), C!=D, len(X), T=1..n-X+1. %(M)
m(T,L) :- r { m(K,T,L) : K=1..r }, len(X), L = 1..X, T=1..n-X+1. %(M')
:- { m(T,L) } 0, len(X), T=1..n-X+1. %(CONS)
#show x/2.
```

8.2 Shortest Minimal Unique Substring (MUS)

To find a unique substring of length $x \in [1..n]$, we slightly modify the above SAT formulation by representing X differently as follows. We first select a starting position $p \in [1..n]$ of X with n Boolean variables p_i with only one set that marks the start of X in $\widetilde{T}$, i.e.,

$$\sum_{i=1}^{n} p_i = 1. \tag{POSP}$$

Each character of X can be chosen such that $X[\ell] \in \widetilde{T}[p + \ell - 1]$ for every $\ell \in [1..x]$. For that we create a two-dimensional table $X'[\ell, k]$ such that $X'[\ell, k] = 1$ if and only if $X[\ell] = \widetilde{T}[p + \ell - 1][k]$. This can be modelled by the following constraint.

$$\forall \ell \in [1..x] : \sum_{k=1}^{|\widetilde{T}[p+\ell-1]|} X'[\ell, k] = 1. \tag{SELX}$$

$\{\mathcal{O}(x), \mathcal{O}(r)\}$

Then we define M and M' as above but omit in Eq. (CONS) the starting position p of X in the range for $t \in [1..n - x + 1] \setminus \{p\}$. The asymptotic complexity of the generated CNF compared to Section 8.1 is that the number of selectable variables is $2n + rx$, which can be

fewer for large $x \in \Theta(n)$ and small r. Listing 2 shows an encoding in ASP. An evaluation on a randomly generated string is shown in Table 1, and can be reevaluated with the source code available on GitHub `https://github.com/koeppl/edstringcharacteristics`. The evaluation ran on an Intel Xeon Gold 6330 CPU with Ubuntu 22.04 and clingo version 5.7.1 for interpreting the ASP encoding.

Listing 2 ASP encoding computing a MUS.

```
1 { len(X) : X = 1..n } 1. %(LENX)
#minimize { X : len(X) }. %(MINX)
1 { pos(P) : P = 1..n } 1. %(POSP)
1 { x(L,C) : t(K,P+L-1,C) } 1 :- len(X), pos(P), L = 1..X. %(SELX)
m(K,T,L) :- t(K,T+L-1,C), x(L,D), C != D, len(X), T=1..n-X+1. %(M)
m(T,L) :- r { m(K,T,L) : K=1..r }, len(X), L = 1..X, T=1..n-X+1. %(M')
:- { m(T,L) } 0, len(X), pos(P), T=1..n-X+1, T != P. %(CONS)
#show x/2.
```

9 Conclusion

We have studied various problems concerning the characterization of ED strings. While notably longest common extension queries and the computation of the longest repeating factor can be done in quadratic time, other problems are NP-hard to compute for specific generalizations of strings such as indeterminate strings. Since every indeterminate string is a GD string, and every GD string is also an ED string, all hardness results are also true for taking ED strings as an input. For future work, it is natural to study whether we can obtain the same results for smaller values of σ and/or r and in some cases for smaller subclasses of ED strings, e.g., indeterminate strings. Additionally to the inequality problem of Appendix C, we would like to study whether some problems can be further parameterized to obtain FPT algorithms, which hopefully work well in practice.

References

1 Paniz Abedin, Arnab Ganguly, Solon P. Pissis, and Sharma V. Thankachan. Range shortest unique substring queries. In *Proc. SPIRE*, volume 11811 of *LNCS*, pages 258–266, 2019. doi:10.1007/978-3-030-32686-9_18.

2 Paniz Abedin, Arnab Ganguly, Solon P. Pissis, and Sharma V. Thankachan. Efficient data structures for range shortest unique substring queries. *Algorithms*, 13(11):276, 2020. doi:10.3390/a13110276.

3 Paniz Abedin, M. Oguzhan Külekci, and Sharma V. Thankachan. A survey on shortest unique substring queries. *Algorithms*, 13(9):224, 2020. doi:10.3390/A13090224.

4 Boran Adas, Ersin Bayraktar, Simone Faro, Ibraheem Elsayed Moustafa, and M. Oguzhan Külekci. Nucleotide sequence alignment and compression via shortest unique substring. In *Proc. IWBBIO*, volume 9044 of *LNCS*, pages 363–374, 2015. doi:10.1007/978-3-319-16480-9_36.

5 Tooru Akagi, Yuki Kuhara, Takuya Mieno, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Combinatorics of minimal absent words for a sliding window. *Theor. Comput. Sci.*, 927:109–119, 2022. doi:10.1016/J.TCS.2022.06.002.

6 Tooru Akagi, Kouta Okabe, Takuya Mieno, Yuto Nakashima, and Shunsuke Inenaga. Minimal absent words on run-length encoded strings. In *Proc. CPM*, volume 223 of *LIPIcs*, pages 27:1–27:17, 2022. doi:10.4230/LIPIcs.CPM.2022.27.

7 Ali Alatabbi, M. Sohel Rahman, and William F. Smyth. Inferring an indeterminate string from a prefix graph. *J. Discrete Algorithms*, 32:6–13, 2015. doi:10.1016/J.JDA.2014.12.006.

8 Daniel R. Allen, Sharma V. Thankachan, and Bojian Xu. An ultra-fast and parallelizable algorithm for finding k-mismatch shortest unique substrings. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 18(1):138–148, 2021. doi:10.1109/TCBB.2020.2968531.

9 Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Degenerate string comparison and applications. In *Proc. WABI*, volume 113 of *LIPIcs*, pages 21:1–21:14, 2018. doi:10.4230/LIPIcs.WABI.2018.21.

10 Mai Alzamel, Alessio Conte, Daniele Greco, Veronica Guerrini, Costas S. Iliopoulos, Nadia Pisanti, Nicola Prezza, Giulia Punzi, and Giovanna Rosone. Online algorithms on antipowers and antiperiods. In *Proc. SPIRE*, volume 11811 of *LNCS*, pages 175–188, 2019. doi:10.1007/978-3-030-32686-9_13.

11 Mai Alzamel, Christopher Hampson, Costas S. Iliopoulos, Zara Lim, Solon P. Pissis, Dimitrios Vlachakis, and Steven Watts. Maximal degenerate palindromes with gaps and mismatches. *Theor. Comput. Sci.*, 978:114182, 2023. doi:10.1016/J.TCS.2023.114182.

12 Pavlos Antoniou, Maxime Crochemore, Costas S. Iliopoulos, Inuka Jayasekera, and Gad M. Landau. Conservative string covering of indeterminate strings. In *Proc. PSC*, pages 108–115, 2008. URL: http://www.stringology.org/event/2008/p10.html.

13 Kotaro Aoyama, Yuto Nakashima, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster online elastic degenerate string matching. In *Proc. CPM*, volume 105 of *LIPIcs*, pages 9:1–9:10, 2018. doi:10.4230/LIPIcs.CPM.2018.9.

14 Rocco Ascone, Giulia Bernardini, Alessio Conte, Massimo Equi, Estéban Gabory, Roberto Grossi, and Nadia Pisanti. A unifying taxonomy of pattern matching in degenerate strings and founder graphs. In *Proc. WABI*, volume 312 of *LIPIcs*, pages 14:1–14:21, 2024. doi:10.4230/LIPIcs.WABI.2024.14.

15 Aqil M. Azmi. On identifying minimal absent and unique words: An efficient scheme. *Cogn. Comput.*, 8(4):603–613, 2016. doi:10.1007/S12559-016-9385-9.

16 Golnaz Badkobeh, Gabriele Fici, and Simon J. Puglisi. Algorithms for anti-powers in strings. *Inf. Process. Lett.*, 137:57–60, 2018. doi:10.1016/j.ipl.2018.05.003.

17 Hideo Bannai, Travis Gagie, Gary Hoppenworth, Simon J. Puglisi, and Luís M. S. Russo. More time-space tradeoffs for finding a shortest unique substring. *Algorithms*, 13(9):234, 2020. doi:10.3390/A13090234.

18 Md. Faizul Bari, Mohammad Sohel Rahman, and Rifat Shahriyar. Finding all covers of an indeterminate string in $o(n)$ time on average. In *Proc. PSC*, pages 263–271, 2009. URL: http://www.stringology.org/event/2009/p24.html.

19 Carl Barton, Alice Héliou, Laurent Mouchard, and Solon P. Pissis. Linear-time computation of minimal absent words using suffix array. *BMC Bioinform.*, 15:388, 2014. doi:10.1186/s12859-014-0388-9.

20 Carl Barton, Alice Héliou, Laurent Mouchard, and Solon P. Pissis. Parallelising the computation of minimal absent words. In *Proc. PPAM*, volume 9574 of *LNCS*, pages 243–253, 2015. doi:10.1007/978-3-319-32152-3_23.

21 Verónica Becher, Alejandro Deymonnaz, and Pablo Ariel Heiber. Efficient computation of all perfect repeats in genomic sequences of up to half a gigabyte, with a case study on the human genome. *Bioinform.*, 25(14):1746–1753, 2009. doi:10.1093/BIOINFORMATICS/BTP321.

22 Timo Beller, Katharina Berger, and Enno Ohlebusch. Space-efficient computation of maximal and supermaximal repeats in genome sequences. In *Proc. SPIRE*, volume 7608 of *LNCS*, pages 99–110, 2012. doi:10.1007/978-3-642-34109-0_11.

23 Aaron Berger and Colin Defant. On anti-powers in aperiodic recurrent words. *Adv. Appl. Math.*, 121:102104, 2020. doi:10.1016/J.AAM.2020.102104.

24 Giulia Bernardini, Estéban Gabory, Solon P. Pissis, Leen Stougie, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string matching with 1 error. In *Proc. LATIN*, volume 13568 of *LNCS*, pages 20–37, 2022. doi:10.1007/978-3-031-20624-5_2.

25 Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Even faster elastic-degenerate string matching via fast matrix multiplication. In *Proc. ICALP*, volume 132 of *LIPIcs*, pages 21:1–21:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.21.

26 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.*, 812:109–122, 2020. doi:10.1016/J.TCS.2019.08.012.

27 Philip Bille, Inge Li Gørtz, and Tord Stordalen. Rank and select on degenerate strings. In *Proc. DCC*, pages 283–292, 2024. doi:10.1109/DCC58796.2024.00036.

28 Francine Blanchet-Sadri, Michelle Bodnar, and Benjamin De Winkle. New bounds and extended relations between prefix arrays, border arrays, undirected graphs, and indeterminate strings. *Theory Comput. Syst.*, 60(3):473–497, 2017. doi:10.1007/S00224-016-9668-2.

29 Thomas Büchler, Jannik Olbrich, and Enno Ohlebusch. Efficient short read mapping to a pangenome that is represented by a graph of ED strings. *Bioinform.*, 39(5), 2023. doi:10.1093/BIOINFORMATICS/BTAD320.

30 Amanda Burcroff. (k, λ)-anti-powers and other patterns in words. *Electron. J. Comb.*, 25(4):4, 2018. doi:10.37236/8032.

31 Supaporn Chairungsee. A new approach for phylogenetic tree construction based on minimal absent words. In *Proc. DEXA*, pages 15–19, 2014. doi:10.1109/DEXA.2014.21.

32 Supaporn Chairungsee and Thana Charuphanthuset. An approach for LPF table computation. In *Proc. DEXA*, volume 1062 of *Communications in Computer and Information Science*, pages 3–7, 2019. doi:10.1007/978-3-030-27684-3_1.

33 Supaporn Chairungsee and Maxime Crochemore. Using minimal absent words to build phylogeny. *Theor. Comput. Sci.*, 450:109–116, 2012. doi:10.1016/j.tcs.2012.04.031.

34 Manolis Christodoulakis, Patrick J. Ryan, William F. Smyth, and Shu Wang. Indeterminate strings, prefix arrays & undirected graphs. *Theor. Comput. Sci.*, 600:34–48, 2015. doi:10.1016/J.TCS.2015.06.056.

35 Aleksander Cislak, Szymon Grabowski, and Jan Holub. SOPanG: online text searching over a pan-genome. *Bioinform.*, 34(24):4290–4292, 2018. doi:10.1093/BIOINFORMATICS/BTY506.

36 Tim Crawford, Golnaz Badkobeh, and David Lewis. Searching page-images of early music scanned with OMR: A scalable solution using minimal absent words. In *Proc. ISMIR*, pages 233–239, 2018. URL: http://ismir2018.ircam.fr/doc/pdfs/210_Paper.pdf.

37 Maxime Crochemore, Gabriele Fici, Robert Mercas, and Solon P. Pissis. Linear-time sequence comparison using minimal absent words & applications. In *Proc. LATIN*, volume 9644 of *LNCS*, pages 334–346, 2016. doi:10.1007/978-3-662-49529-2_25.

38 Maxime Crochemore, Alice Héliou, Gregory Kucherov, Laurent Mouchard, Solon P. Pissis, and Yann Ramusat. Minimal absent words in a sliding window and applications to on-line pattern matching. In *Proc. FCT*, volume 10472 of *LNCS*, pages 164–176, 2017. doi:10.1007/978-3-662-55751-8_14.

39 Maxime Crochemore and Lucian Ilie. Computing longest previous factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008. doi:10.1016/j.ipl.2007.10.006.

40 Maxime Crochemore, Lucian Ilie, Costas S. Iliopoulos, Marcin Kubica, Wojciech Rytter, and Tomasz Walen. LPF computation revisited. In *Proc. IWOCA*, volume 5874 of *LNCS*, pages 158–169, 2009. doi:10.1007/978-3-642-10217-2_18.

41 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Covering problems for partial words and for indeterminate strings. *Theor. Comput. Sci.*, 698:25–39, 2017. doi:10.1016/J.TCS.2017.05.026.

42 Jacqueline W. Daykin and Bruce W. Watson. Indeterminate string factorizations and degenerate text transformations. *Math. Comput. Sci.*, 11(2):209–218, 2017. doi:10.1007/S11786-016-0285-X.

43 Colin Defant. Anti-power prefixes of the thue–morse word. *Electron. J. Comb.*, 24(1):1, 2017. doi:10.37236/6321.

44 Gabriele Fici and Pawel Gawrychowski. Minimal absent words in rooted and unrooted trees. In *Proc. SPIRE*, volume 11811 of *LNCS*, pages 152–161, 2019. doi:10.1007/978-3-030-32686-9_11.

45 Gabriele Fici, Mickaël Postic, and Manuel Silva. Abelian antipowers in infinite words. *Adv. Appl. Math.*, 108:67–78, 2019. doi:10.1016/j.aam.2019.04.001.

46 Gabriele Fici, Antonio Restivo, Manuel Silva, and Luca Q. Zamboni. Anti-powers in infinite words. *J. Comb. Theory, Ser. A*, 157:109–119, 2018. doi:10.1016/j.jcta.2018.02.009.

47 Frantisek Franek, Jan Holub, William F. Smyth, and Xiangdong Xiao. Computing quasi suffix arrays. *Journal of Automata, Languages and Combinatorics*, 8(4):593–606, 2003. doi:10.25596/jalc-2003-593.

48 Yuta Fujishige, Takuya Takagi, and Diptarama Hendrian. Truncated DAWGs and their application to minimal absent word problem. In *Proc. SPIRE*, volume 11147 of *LNCS*, pages 139–152, 2018. doi:10.1007/978-3-030-00479-8_12.

49 Yuta Fujishige, Yuki Tsujimaru, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing DAWGs and minimal absent words in linear time for integer alphabets. In *Proc. MFCS*, volume 58 of *LIPIcs*, pages 38:1–38:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.38.

50 Estéban Gabory, Njagi Moses Mwaniki, Nadia Pisanti, Solon P. Pissis, Jakub Radoszewski, Michelle Sweering, and Wiktor Zuba. Comparing elastic-degenerate strings: Algorithms, lower bounds, and applications. In *Proc. CPM*, volume 259 of *LIPIcs*, pages 11:1–11:20, 2023. doi:10.4230/LIPIcs.CPM.2023.11.

51 Estéban Gabory, Njagi Moses Mwaniki, Nadia Pisanti, Solon P. Pissis, Jakub Radoszewski, Michelle Sweering, and Wiktor Zuba. Pangenome comparison via ED strings. *Frontiers Bioinform.*, 4, 2024. doi:10.3389/FBINF.2024.1397036.

52 Estéban Gabory, Njagi Moses Mwaniki, Nadia Pisanti, Solon P. Pissis, Jakub Radoszewski, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string comparison. *Inf. Comput.*, 304:105296, 2025. doi:10.1016/J.IC.2025.105296.

53 Estéban Gabory, Eric Rivals, Michelle Sweering, Hilde Verbeek, and Pengfei Wang. Periodicity of degenerate strings. In *Proc. PSC*, pages 42–56, 2023. URL: http://www.stringology.org/event/2023/p05.html.

54 Marisa R. Gaetz. Anti-power j-fixes of the Thue–Morse word. *Discret. Math. Theor. Comput. Sci.*, 23(1), 2021. doi:10.46298/DMTCS.5483.

55 Arnab Ganguly, Wing-Kai Hon, Rahul Shah, and Sharma V. Thankachan. Space-time trade-offs for finding shortest unique substrings and maximal unique matches. *Theor. Comput. Sci.*, 700:75–88, 2017. doi:10.1016/j.tcs.2017.08.002.

56 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. A Series of books in the mathematical sciences. Bell Laboratories, 1979.

57 Pawel Gawrychowski, Samah Ghazawi, and Gad M. Landau. On indeterminate strings matching. In *Proc. CPM*, volume 161 of *LIPIcs*, pages 14:1–14:14, 2020. doi:10.4230/LIPIcs.CPM.2020.14.

58 Pawel Gawrychowski, Adam Górkiewicz, Pola Marciniak, Solon P. Pissis, and Karol Pokorski. Faster approximate elastic-degenerate string matching - part B. In *Proc. CPM*, volume 331 of *LIPIcs*, pages 29:1–29:21, 2025. doi:10.4230/LIPIcs.CPM.2025.29.

59 Daniel Gibney. An efficient elastic-degenerate text index? not likely. In *Proc. SPIRE*, volume 12303 of *LNCS*, pages 76–88, 2020. doi:10.1007/978-3-030-59212-7_6.

60 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

61 Bernhard Haubold, Nora Pierstorff, Friedrich Möller, and Thomas Wiehe. Genome comparison without alignment using shortest unique substrings. *BMC Bioinform.*, 6:123, 2005. doi:10.1186/1471-2105-6-123.

62 Alice Héliou, Solon P. Pissis, and Simon J. Puglisi. emMAW: computing minimal absent words in external memory. *Bioinform.*, 33(17):2746–2749, 2017. doi:10.1093/BIOINFORMATICS/BTX209.

63 Joel Helling, Patrick J. Ryan, W. F. Smyth, and Michael Soltys. Constructing an indeterminate string from its associated graph. *Theor. Comput. Sci.*, 710:88–96, 2018. doi:10.1016/J.TCS.2017.02.016.

64 Jan Holub, William F. Smyth, and Shu Wang. Fast pattern-matching on indeterminate strings. *J. Discrete Algorithms*, 6(1):37–50, 2008. doi:10.1016/J.JDA.2006.10.003.

65 Wing-Kai Hon, Sharma V. Thankachan, and Bojian Xu. In-place algorithms for exact and approximate shortest unique substring problems. *Theor. Comput. Sci.*, 690:12–25, 2017. doi:10.1016/J.TCS.2017.05.032.

66 John E. Hopcroft and R. M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report 71–114, University of California, 1971.

67 Xiaocheng Hu, Jian Pei, and Yufei Tao. Shortest unique queries on strings. In *Proc. SPIRE*, volume 8799 of *LNCS*, pages 161–172, 2014. doi:10.1007/978-3-319-11918-2_16.

68 Harry B. Hunt III, Daniel J. Rosenkrantz, and Thomas G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *J. Comput. Syst. Sci.*, 12(2):222–268, 1976. doi:10.1016/S0022-0000(76)80038-4.

69 Atalay Mert Ileri, M. Oguzhan Külekci, and Bojian Xu. A simple yet time-optimal and linear-space algorithm for shortest unique substring queries. *Theor. Comput. Sci.*, 562:621–633, 2015. doi:10.1016/J.TCS.2014.11.004.

70 Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate strings. *Inf. Comput.*, 279:104616, 2021. doi:10.1016/J.IC.2020.104616.

71 IUPAC-IUB Commission on Biochemical Nomenclature (CBN). Abbreviations and symbols for nucleic acids, polynucleotides and their constituents. *Journal of Molecular Biology*, 55(3):299–310, 1971. doi:10.1016/0022-2836(71)90319-6.

72 Richard M. Karp. Reducibility among combinatorial problems. In *Proc. Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

73 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Efficient representation and counting of antipower factors in words. *Inf. Comput.*, 286:104779, 2022. doi:10.1016/J.IC.2021.104779.

74 M. Oguzhan Külekci, Jeffrey Scott Vitter, and Bojian Xu. Efficient maximal repeat finding using the Burrows–Wheeler transform and wavelet tree. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 9(2):421–429, 2012. doi:10.1109/TCBB.2011.127.

75 Wen-Wei Liao, Mobin Asri, Jana Ebler, Daniel Doerr, Marina Haukness, Glenn Hickey, Shuangjia Lu, Julian K. Lucas, Jean Monlong, Haley J. Abel, Silvia Buonaiuto, Xian H. Chang, Haoyu Cheng, Justin Chu, Vincenza Colonna, Jordan M. Eizenga, Xiaowen Feng, Christian Fischer, Robert S. Fulton, Shilpa Garg, Cristian Groza, Andrea Guarracino, William T. Harvey, Simon Heumos, Kerstin Howe, Miten Jain, Tsung-Yu Lu, Charles Markello, Fergal J. Martin, Matthew W. Mitchell, Katherine M. Munson, Moses Njagi Mwaniki, Adam M. Novak, Hugh E. Olsen, Trevor Pesout, David Porubsky, Pjotr Prins, Jonas A. Sibbesen, Jouni Sirén, Chad Tomlinson, Flavia Villani, Mitchell R. Vollger, Lucinda L. Antonacci-Fulton, Gunjan Baid, Carl A. Baker, Anastasiya Belyaeva, Konstantinos Billis, Andrew Carroll, Pi-Chuan Chang, Sarah Cody, Daniel E. Cook, Robert M. Cook-Deegan, Omar E. Cornejo, Mark Diekhans, Peter Ebert, Susan Fairley, Olivier Fedrigo, Adam L. Felsenfeld, Giulio Formenti, Adam Frankish, Yan Gao, Nanibaa' A. Garrison, Carlos Garcia Giron, Richard E. Green, Leanne Haggerty, Kendra Hoekzema, Thibaut Hourlier, Hanlee P. Ji, Eimear E. Kenny, Barbara A. Koenig, Alexey Kolesnikov, Jan O. Korbel, Jennifer Kordosky, Sergey Koren, HoJoon Lee, Alexandra P. Lewis, Hugo Magalhães, Santiago Marco-Sola, Pierre Marijon, Ann McCartney, Jennifer McDaniel, Jacquelyn Mountcastle, Maria Nattestad, Sergey Nurk, Nathan D. Olson, Alice B. Popejoy, Daniela Puiu, Mikko Rautiainen, Allison A. Regier, Arang Rhie, Samuel Sacco, Ashley D. Sanders, Valerie A. Schneider, Baergen I. Schultz, Kishwar Shafin, Michael W. Smith, Heidi J. Sofia, Ahmad N. Abou Tayoun, Françoise Thibaud-Nissen, Francesca Floriana Tricomi, Justin Wagner, Brian Walenz, Jonathan M. D. Wood, Aleksey V. Zimin, Guillaume

Bourque, Mark J. P. Chaisson, Paul Flicek, Adam M. Phillippy, Justin M. Zook, Evan E. Eichler, David Haussler, Ting Wang, Erich D. Jarvis, Karen H. Miga, Erik Garrison, Tobias Marschall, Ira M. Hall, Heng Li, and Benedict Paten. A draft human pangenome reference. *Nature*, 617(7960):312–324, 2023. doi:10.1038/s41586-023-05896-x.

76 Felipe A. Louza, Neerja Mhaskar, and W. F. Smyth. A new approach to regular & indeterminate strings. *Theor. Comput. Sci.*, 854:105–115, 2021. doi:10.1016/j.tcs.2020.12.007.

77 Sorina Maciuca, Carlos del Ojo Elias, Gil McVean, and Zamin Iqbal. A natural encoding of genetic variation in a Burrows–Wheeler transform to enable mapping and genome inference. In *Proc. WABI*, volume 9838 of *LNCS*, pages 222–233, 2016. doi:10.1007/978-3-319-43681-4_18.

78 David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, 1978. doi:10.1145/322063.322075.

79 Takuya Mieno, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Shortest unique substring queries on run-length encoded strings. In *Proc. MFCS*, volume 58 of *LIPIcs*, pages 69:1–69:11, 2016. doi:10.4230/LIPIcs.MFCS.2016.69.

80 Takuya Mieno, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Tight bounds on the maximum number of shortest unique substrings. In *Proc. CPM*, volume 78 of *LIPIcs*, pages 24:1–24:11, 2017. doi:10.4230/LIPIcs.CPM.2017.24.

81 Takuya Mieno, Dominik Köppl, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Space-efficient algorithms for computing minimal/shortest unique substrings. *Theor. Comput. Sci.*, 845:230–242, 2020. doi:10.1016/j.tcs.2020.09.017.

82 Takuya Mieno, Yuki Kuhara, Tooru Akagi, Yuta Fujishige, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Minimal unique substrings and minimal absent words in a sliding window. In *Proc. SOFSEM*, volume 12011 of *LNCS*, pages 148–160, 2020. doi:10.1007/978-3-030-38919-2_13.

83 Njagi Moses Mwaniki, Erik Garrison, and Nadia Pisanti. Fast exact string to d-texts alignments. In *Proc. BIOSTEC*, pages 70–79, 2023. doi:10.5220/0011666900003414.

84 Njagi Moses Mwaniki and Nadia Pisanti. Optimal sequence alignment to ED-strings. In *Proc. ISBRA*, volume 13760 of *LNCS*, pages 204–216, 2022. doi:10.1007/978-3-031-23198-8_19.

85 Sumaiya Nazeen, M. Sohel Rahman, and Rezwana Reaz. Indeterminate string inference algorithms. *J. Discrete Algorithms*, 10:23–34, 2012. doi:10.1016/J.JDA.2011.08.002.

86 Mhaskar Neerja and William F. Smyth. Simple KMP pattern-matching on indeterminate strings. In *Proc. PSC*, pages 125–133, 2020. URL: http://www.stringology.org/event/2020/p11.html.

87 Enno Ohlebusch and Stefan Kurtz. Space efficient computation of rare maximal exact matches between multiple sequences. *J. Comput. Biol.*, 15(4):357–377, 2008. doi:10.1089/CMB.2007.0105.

88 Kouta Okabe, Takuya Mieno, Yuto Nakashima, Shunsuke Inenaga, and Hideo Bannai. Linear-time computation of generalized minimal absent words for multiple strings. In *Proc. SPIRE*, volume 14240 of *LNCS*, pages 331–344, 2023. doi:10.1007/978-3-031-43980-3_27.

89 Takahiro Ota and Akiko Manada. A reconstruction of circular binary string using substrings and minimal absent words. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 107(3):409–416, 2024. doi:10.1587/TRANSFUN.2023TAP0015.

90 Jian Pei, Wush Chi-Hsuan Wu, and Mi-Yen Yeh. On shortest unique substring queries. In *Proc. ICDE*, pages 937–948, 2013. doi:10.1109/ICDE.2013.6544887.

91 Armando J. Pinho, Paulo Jorge S. G. Ferreira, Sara P. Garcia, and João M. O. S. Rodrigues. On finding minimal absent words. *BMC Bioinform.*, 10, 2009. doi:10.1186/1471-2105-10-137.

92 Solon P. Pissis, Jakub Radoszewski, and Wiktor Zuba. Faster approximate elastic-degenerate string matching - part A. In *Proc. CPM*, volume 331 of *LIPIcs*, pages 28:1–28:19, 2025. doi:10.4230/LIPIcs.CPM.2025.28.

93 Petr Procházka, Ondrej Cvacho, Lubos Krcál, and Jan Holub. Backward pattern matching on elastic-degenerate strings. *SN Comput. Sci.*, 4(5):442, 2023. doi:10.1007/S42979-023-01760-X.

94 Luís M. S. Russo, Diogo M. Costa, Rui Henriques, Hideo Bannai, and Alexandre P. Francisco. Order-preserving pattern matching indeterminate strings. *Inf. Comput.*, 289(Part):104924, 2022. doi:10.1016/J.IC.2022.104924.

95 Daniel W. Schultz and Bojian Xu. Parallel methods for finding k-mismatch shortest unique substrings using GPU. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 18(1):386–395, 2021. doi:10.1109/TCBB.2019.2935061.

96 William F. Smyth and Shu Wang. An adaptive hybrid pattern-matching algorithm on indeterminate strings. *Int. J. Found. Comput. Sci.*, 20(6):985–1004, 2009. doi:10.1142/S0129054109007005.

97 Yun Tian and Bojian Xu. On longest repeat queries using GPU. In *Proc. DASFAA*, volume 9049 of *LNCS*, pages 316–333, 2015. doi:10.1007/978-3-319-18120-2_19.

98 Kazuya Tsuruta, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Shortest unique substrings queries in optimal time. In *Proc. SOFSEM*, volume 8327 of *LNCS*, pages 503–513, 2014. doi:10.1007/978-3-319-04298-5_44.

99 Bojian Xu. On stabbing queries for generalised longest repeats. *Int. J. Data Min. Bioinform.*, 15(4):350–371, 2016. doi:10.1504/IJDMB.2016.078152.

A Longest Repeating Substring

The *longest repeating substring (LRS)* of a classic string T is the longest substring that occurs at least twice in T. The search for longest repeating substring is important, among others, for detecting similarities in biological sequences [21]. While textbook solutions for this problem exist [60, APL4], the problem has been extended such that the reported substring has to cover specific string positions [97, 99] or needs to be common among several input texts [87]. While the textbook solution uses suffix trees, ideas have been adopted to BWT-based indexes [22, 74].

We generalize LRS to ED strings as follows.

▶ **Problem 6** (LRF problem). *Given an ED string $\widetilde{S}$, the LRF problem is to determine a longest string that occurs at least twice in the language of $\widetilde{S}$ but has to start at least at two different text-positions in $\widetilde{S}$.*

Our idea to compute LRF is to reduce the problem to longest common extension (LCE) queries. The LCE query in a classic string T asks for the longest common prefix of two suffixes $T[i..]$ and $T[j..]$. In the ED string setting, an LCE query at two text-positions asks for the longest string that has an occurrence starting at both text-positions.

Given an ED string $\widetilde{S}$ with size $N = ||\widetilde{S}||$, to find an LRS, we compute the LCE of every pair of pairwise different text-positions, and report the longest LCE length we discovered.

We note that we can directly leverage the longest common substring solution of [52, Section 6.4] and [51] working in $O(Nm)$ time, where $m = \sum_{i=1} n|\widetilde{S}[i]| \leq N$ is the sum of the cardinalities of the ED symbols of $\widetilde{S}$. There, the authors reduce their problem to LCE queries on distinct positions, from which we can extract our solution by selecting the maximum computed LCE value among pairs of distinct positions. Nevertheless, we present here a less sophisticated but straight-forward solution that matches this time bounds when $m = \Theta(n)$.

Our idea is that we can compute all LCE lengths in $\mathcal{O}(N^2)$ time with dynamic programming by using the linearized form of an ED string [29], which maps an ED string to a classic string whose alphabet contains three additional special characters. The linearized form of an ED symbol $\{W_1, W_2, \ldots, W_k\}$ is $(W_1|W_2|\ldots|W_k) \in (\Sigma \cup \{(,),|\})^*$. The

characters in $\{\texttt{(},\texttt{)},\texttt{|}\}$ are called *special* and are assumed to be not in Σ. The linearized form of $\left\{ W_1 \right\}$ is just W_1. Further we stipulate that ε is always the last string in an ED symbol if it is present. Then the linearized form of an ED string $\widetilde{S}$, denoted by L, is the concatenation of the linearized forms of its symbols. For example, the linearized form of $\widetilde{S} = \texttt{b}\left\{ \begin{smallmatrix} \texttt{a} \\ \varepsilon \end{smallmatrix} \right\}\texttt{c}\left\{ \begin{smallmatrix} \texttt{abc} \\ \texttt{c} \end{smallmatrix} \right\}\left\{ \begin{smallmatrix} \texttt{a} \\ \texttt{b} \end{smallmatrix} \right\}$ is $L = \texttt{b(a|)c(abc|c)(a|b)}$. Since the number of characters to write increases by a multiplicative constant[1], $|L| \in \mathcal{O}(N)$. The next function $\mathrm{next}(i, c) = \min\{j \mid j > i, L[j] = c\} \cup \{1 + |L|\}$ gives the first text position of an occurrence of c in $L[i+1..]$, or otherwise the invalid position $1 + |L|$. Let

$$\mathcal{P}(i) := \{i+1\} \cup \{j+1 \mid i < j < \mathrm{next}(i, \texttt{')'}) \wedge L[j] = \texttt{'|'}\} \text{ for } L[i] = \texttt{'('}$$

denote all starting positions of non-special character strings within the parentheses starting at $L[i]$ divided by `'|'` characters, or `')'` in case that the last string is ε. For instance, $\mathcal{P}(7) = \{8, 12\}$ and $\mathcal{P}(2) = \{3, 5\}$ for our example. Then the function adj below gives the set of starting positions of non-special characters we want to match in case one of the characters is special.

$$\mathrm{adj}(i) = \begin{cases} \{i\} & \text{if } L[i] \in \Sigma, \\ \{\mathrm{next}(i, \texttt{')'}) + 1\} & \text{if } L[i] = \texttt{'|'}, \\ \{i+1\} & \text{if } L[i] = \texttt{')'}, \\ \mathcal{P}(i) & \text{if } L[i] = \texttt{'('}. \end{cases}$$

In particular, $\mathrm{adj}(i)$ advances if and only if i is a position of a special character. We can precompute $\mathrm{adj}(i)$ for all i in linear time by scanning L from right to left with a stack maintaining the positions of the recently read special characters.

Our goal is to compute a DP table for the longest common extension of two text-positions in $\widetilde{S}$ that are mapped to the indices i and j in L (the mapping of text-positions to L-positions is injective but not surjective). We obtain this table by additionally specifying values for special character positions in L, using the following recursion formula.

$$D(i,j) = \begin{cases} 0 & \text{if } \max\{i,j\} > |L|, \\ 1 + D(i+1, j+1) & \text{if } L[i] = L[j] \text{ and } L[i] \in \Sigma, \\ 0 & \text{if } L[i] \neq L[j] \text{ and } L[i], L[j] \in \Sigma, \\ \max\{D(i', j') \mid (i', j') \in \mathrm{adj}(i) \times \mathrm{adj}(j)\} & \text{otherwise.} \end{cases}$$

▶ **Example 13.** We evaluate $D(1, 9)$ of the above example. We visualize L with indices.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$L[i] =$	b	(	a	|	)	c	(	a	b	c	|	c	)	(	a	|	b	)

To calculate $D(1, 9)$, we recursively apply the above formula.

- $D(1, 9) = 1 + D(2, 10)$
- $D(2, 10) = \max\{D(3, 10), D(5, 10)\}$
  $= D(6, 10)$ because $D(3, 10) = 0$ and $D(5, 10) = D(6, 10)$
- $D(6, 10) = 1 + D(7, 11)$

[1] The worst case is $\widetilde{S} = \left\{ \begin{smallmatrix} a \\ b \end{smallmatrix} \right\}\left\{ \begin{smallmatrix} a \\ b \end{smallmatrix} \right\}\cdots\left\{ \begin{smallmatrix} a \\ b \end{smallmatrix} \right\}$", for which the size grows from $||\widetilde{S}|| = 2n$ to $|L| = 5n$ by a factor of 2.5.

- $D(7,11) = \max\{D(8,14), D(12,14)\}$
- $D(8,14) = \max\{D(8,15), D(8,17)\} = 1$
- $D(12,14) = \max\{D(12,15), D(12,17)\} = 0$

We used that $D(12,15) = 0$, $D(12,17) = 0$, $D(8,15) = 1$, and $D(8,17) = 0$. Hence, $D(1,9) = 1 + D(2,10) = 1 + D(6,10) = 1 + 1 + D(7,11) = 2 + D(8,15) = 3$.

We can compute $D(i,j)$ in constant time when neither $L[i]$ nor $L[j]$ is an open parenthesis character `'('`. Because parentheses are neither nested nor overlapping in L by construction, $\mathrm{adj}(i) \cap \mathrm{adj}(j) = \emptyset$ for all $i \neq j$ with $L[i] = L[j] =$ `'('`. Therefore, each pair of positions (i', j') is included in at most one computation of $D(i,j)$ with $L[i] = L[j] =$ `'('`. Similarly, we use each $\mathrm{adj}(i)$ (with $L[i] =$ `'('`) in at most $\mathcal{O}(N)$ computations of $D(i,j)$ with $L[j] \neq$ `'('`. By this amortization argument, we compute D in $\mathcal{O}(N^2)$ time.

▶ **Theorem 14.** *We can solve* LRF *in* $\mathcal{O}(N^2)$ *time.*

It is also unlikely that we can do better in general. To understand that, we realize that D can be used to solve the ED String Intersection problem (EDSI) [50]: Let $\widetilde{S_1}$ and $\widetilde{S_2}$ be two ED strings and $c \in \Sigma$ a character that has no occurrence in neither $\widetilde{S_1}$ nor $\widetilde{S_2}$. Then the longest repeating substring of $\{ c^k \} \widetilde{S_1} \{ c^k \} \widetilde{S_2} \{ c^k \}$ has length at least $2k$ if and only if $\mathcal{L}(\widetilde{S_1}) \cap \mathcal{L}(\widetilde{S_2}) \neq \emptyset$ for a sufficiently large $k \in \mathcal{O}(N)$ with $N = ||\widetilde{S_1}|| + ||\widetilde{S_2}||$.

▶ **Lemma 15.** *Under the Strong Exponential-Time Hypothesis (SETH), there is no* $\mathcal{O}(N^{2-\varepsilon})$*-time algorithm for* LRF *for any constant* $\varepsilon > 0$.

Proof. [50, Thm. 2] showed that there is no $\mathcal{O}(N^{2-\varepsilon})$-time algorithm for any constant $\varepsilon > 0$ for EDSI under SETH. ◀

B LPF for Indeterminate Strings

We show that Problem 4 can be solved in polynomial time if the input is an indeterminate string $\widetilde{S}$. Let j be the start of the longest previous factor of i, and let L be this LPF. If $|L| < i - j$, there is no overlap and the LPF corresponds to the longest common prefix of $\widetilde{S}[i..]$ and $\widetilde{S}[j..]$. Otherwise, L must be periodic with a period $p \leq i - j$, where p divides $i - j$ – otherwise we would take different paths at $\widetilde{S}[i]$. That is, there is a string U of length p and a (possibly empty) proper prefix V of U such that $L = U^{\lfloor |L|/p \rfloor} \cdot V$.

We have $\min\{\mathrm{LCP}(i,j), \mathrm{LCP}(i+p,j), \mathrm{LCP}(i+2p,j), \ldots, \mathrm{LCP}(i+\lfloor |L|/p - 1 \rfloor p, j)\} \geq p$ and $\mathrm{LCP}(i + \lfloor |L|/p \rfloor p, j) \geq |V|$. Note that $|V|$ can be derived from $|L|$ and p.

The converse is also true, that is, if the above conditions are satisfied, we have an LPF of length $|L|$, because we have these paths of lengths p that we repeatedly take, including at the positions i and j. For each pair (i,j), we can try all periods and possible LPF lengths $|L|$ and pick the longest.

C Hardness of Inequality?

Given two ED strings $\widetilde{S}$ and $\widetilde{T}$, we want to address the problem to decide whether $\mathcal{L}(\widetilde{S}) \neq \mathcal{L}(\widetilde{T})$. If both strings are GD strings, then each symbol stores strings of equal length. It is therefore possible to construct a finite deterministic automaton for each symbol with one starting and accepting state, and connect starting and ending states of subsequent symbols to form an automaton for a GD string $\widetilde{S}$ that accepts $\mathcal{L}(\widetilde{S})$, cf. [9, Sect. 3]. For instance in Figure 2, the state q_4 is the accepting state of the first symbol and the starting state of

the second symbol. Finally, equivalence of two finite deterministic automata can be checked in time linear in their sizes [66]. The same construction for a (regular) ED string[2] may create a finite nondeterministic automaton because the length difference in an ED symbol can cause ε-transitions, cf. Figure 2. Testing inequivalence of two nondeterministic finite automata is PSPACE-complete [56, AL1] while testing inequivalence of regular expressions without Kleene star is NP-complete [56, Problem 3.3.10][68, Thm. 2.7]. Since the latter type of expressions is a superclass of ED strings, we know that inequality testing for ED strings is in NP.

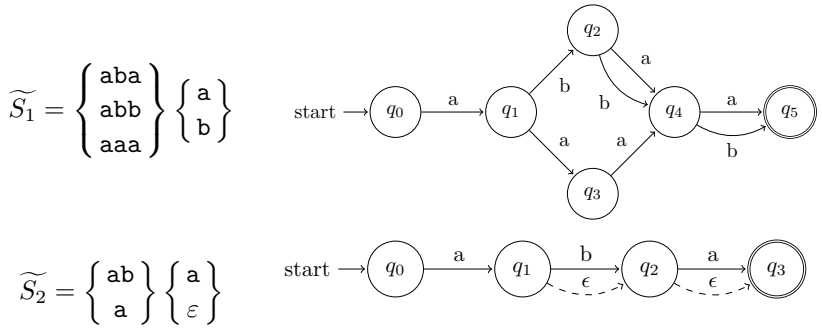


Figure 2 Automata representing the GD string $\widetilde{S_1}$ (top) and the ED string $\widetilde{S_2}$ (bottom). While it is possible to construct an automaton in the size of an ED string, the automaton for ED strings has ε-transitions which may blow up the size exponentially when transforming the nondeterministic automaton to a deterministic one.

Based on the proof of [68, Thm. 2.7], we further can show that inequivalence testing is NP-complete for a slight extension of ED strings allowing one level of nesting, which we call 2ED strings. A *2ED string* is a string $\widetilde{T}$ such that each symbol $\widetilde{T}[i]$ is a set of ED strings. To this end, we reuse the reduction from 3-SAT described Section 4, in particular the family of indeterminate strings $\{\widetilde{T_i}\}_{i=1}^{m}$ constructed in the proof of Theorem 1. We define the two 2ED strings $\widetilde{S}$ and $\widetilde{T}$ to compare with as

$$\widetilde{S} = \left\{ \begin{matrix} \widetilde{T_1} \\ \vdots \\ \widetilde{T_m} \end{matrix} \right\} \text{ and } \widetilde{T} = \left\{ \begin{matrix} 0 \\ 1 \end{matrix} \right\}^n .$$

We observe that $\widetilde{T}$ is a (regular) ED string with $\mathcal{L}(\widetilde{T}) = \{0,1\}^n$, i.e., the language of $\widetilde{T}$ generates all binary strings of length n. Finally, $\mathcal{L}(\widetilde{S}) \neq \mathcal{L}(\widetilde{T})$ if and only if there is a satisfying assignment.

However, the complexity of deciding whether $\mathcal{L}(\widetilde{S}) \neq \mathcal{L}(\widetilde{T})$ for two (regular) ED strings is left open for future work.

[2] In this section, we sometimes use the adjective *regular* to emphasize on the fact that we consider an ED-string, not a specific subclass like an indeterminate string nor a superclass.