# String Sanitization Under Edit Distance

**Giulia Bernardini**
University of Milano - Bicocca, Milan, Italy
giulia.bernardini@unimib.it

**Huiping Chen**
King's College London, UK
huiping.chen@kcl.ac.uk

**Grigorios Loukides**
King's College London, UK
grigorios.loukides@kcl.ac.uk

**Nadia Pisanti**
University of Pisa, Italy
ERABLE Team, Lyon, France
pisanti@di.unipi.it

**Solon P. Pissis**
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands
ERABLE Team, Lyon, France
solon.pissis@cwi.nl

**Leen Stougie**
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands
ERABLE Team, Lyon, France
leen.stougie@cwi.nl

**Michelle Sweering**
CWI, Amsterdam, The Netherlands
michelle.sweering@cwi.nl

## Abstract

Let $W$ be a string of length $n$ over an alphabet $\Sigma$, $k$ be a positive integer, and $\mathcal{S}$ be a set of length-$k$ substrings of $W$. The ETFS problem asks us to construct a string $X_{\text{ED}}$ such that: (i) no string of $\mathcal{S}$ occurs in $X_{\text{ED}}$; (ii) the order of all other length-$k$ substrings over $\Sigma$ is the same in $W$ and in $X_{\text{ED}}$; and (iii) $X_{\text{ED}}$ has minimal edit distance to $W$. When $W$ represents an individual's data and $\mathcal{S}$ represents a set of confidential substrings, algorithms solving ETFS can be applied for utility-preserving string sanitization [Bernardini et al., ECML PKDD 2019]. Our first result here is an algorithm to solve ETFS in $\mathcal{O}(kn^2)$ time, which improves on the state of the art [Bernardini et al., arXiv 2019] by a factor of $|\Sigma|$. Our algorithm is based on a non-trivial modification of the classic dynamic programming algorithm for computing the edit distance between two strings. Notably, we also show that ETFS cannot be solved in $\mathcal{O}(n^{2-\delta})$ time, for any $\delta > 0$, unless the strong exponential time hypothesis is false. To achieve this, we reduce the edit distance problem, which is known to admit the same conditional lower bound [Bringmann and Künnemann, FOCS 2015], to ETFS.

**2012 ACM Subject Classification** Theory of computation → Pattern matching

**Keywords and phrases** String algorithms, data sanitization, edit distance, dynamic programming, conditional lower bound

**Digital Object Identifier** 10.4230/LIPIcs.CPM.2020.7

**Funding** This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 872539.
*Giulia Bernardini*: Partially supported by a research internship at CWI.
*Huiping Chen*: Supported by a Ph.D scholarship from China Scholarship Council.
*Leen Stougie*: Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.
*Michelle Sweering*: Supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

**Acknowledgements** The authors would like to thank Takuya Mieno (Kyushu University) for proofreading the manuscript.

## 1 Introduction

Strings are being used to represent individuals' data arising from a large range of domains e.g., transportation, web analytics, or molecular biology. For example, a string can represent: an individual's movement history, with each letter corresponding to a visited location [23, 20]; an individual's purchasing history in a retailer, with each letter corresponding to a purchased product [9]; or a fragment of the genome sequence of a patient, with each letter corresponding to a DNA base [17]. Analyzing such strings is thus necessary in many different applications. To support these applications, string data must often be disseminated beyond the party that has collected them. For example, transportation organizations disseminate their collected data to data analytics companies [14], retailers disseminate their data to marketing agencies [15], and hospitals disseminate DNA sequencing data to research institutions [17].

However, individuals' data dissemination has led to justified privacy concerns [22] due to the exposure of confidential information [3, 1, 6, 5]. To ease these concerns and comply with legislation such as HIPAA [8] in the US or GDPR [19] in the EU, it is essential to ensure that confidential information does not occur in the disseminated data; this process is called *sanitization*. At the same time, it is essential to preserve the *utility* of the original data, so that data processing and analysis tasks can be accurately performed on the disseminated data. For instance, a data analyst (resp. marketing researcher) should still be able to discover actionable patterns of locations (resp. purchased products) from transportation (resp. purchasing history) data [15, 14].

The *Combinatorial String Dissemination* (CSD) model was recently introduced in [3] to provide privacy and utility guarantees. In CSD, we are given a string $W$ and the aim is to apply a sequence of edit operations to $W$, so that the resulting counterpart $X$ of $W$ satisfies a set of *privacy constraints* and a set of *utility properties*. Specifically, in [3] the authors considered the following CSD problem, referred to as TFS (Total order, Frequency, Sanitization). Given $W$ of length $n$ over an alphabet $\Sigma$, a positive integer $k$, and a set $\mathcal{S}$ of *sensitive* length-$k$ substrings of $W$ modeling confidential information, construct the *minimal-length* string $X$ such that: $X$ does not contain any sensitive length-$k$ substring (**C1**); and the order (and thus the frequency) of all other length-$k$ substrings over $\Sigma$ in $W$ is the same as in $X$ (**P1**). The constraint **C1** ensures that no sensitive length-$k$ substring occurs in $X$. The property **P1** ensures that $X$ incurs minimal utility loss for tasks based on the sequential nature of length-$k$ non-sensitive substrings of $W$, as well as on their frequency. The authors of [3] proposed an $\mathcal{O}(kn)$-time algorithm to solve the TFS problem, and showed that their algorithm is in fact worst-case optimal because the length of $X$ is in $\Theta(kn)$.

The authors of [4] considered another CSD problem related to *edit distance*, that is, the minimum number of insertions, deletions or substitutions of letters needed to transform one string into another. The problem considered in [4] is referred to as ETFS (Edit distance, Total order, Frequency, Sanitization). Given $W$ of length $n$ over an alphabet $\Sigma$, a positive integer $k$, and a set $\mathcal{S}$ of sensitive length-$k$ substrings of $W$, ETFS asks us to construct a string $X_{\mathrm{ED}}$ that satisfies **C1** and **P1** while being at *minimal edit distance* from $W$. ETFS is the main problem we consider in this paper. Strings constructed by means of solving ETFS can be used, with minimal utility loss, in tasks that are based on edit distance as a similarity measure. Examples of such tasks are frequent pattern mining [21], clustering [12], entity extraction [24] and range query answering [16]. To solve ETFS, the authors of [4] proposed an $\mathcal{O}(k|\Sigma|n^2)$-time algorithm. Their algorithm is based on solving a specific instance of approximate regular expression matching, essentially applying the algorithm of Myers and Miller [18] on an appropriate regular expression that models all strings satisfying **C1** and **P1** to finally pick the one that is at minimal edit distance to $W$.

Note that, to have a solution to TFS or ETFS, we may need to insert in $W$ a letter $\# \notin \Sigma$. Indeed, inserting (or replacing letters of $W$ with) any letter of $\Sigma$ could violate **P1** and/or possibly create new occurrences of sensitive length-$k$ substrings. We thus generally have that both $X$ (the solution of TFS) and $X_{\mathrm{ED}}$ (a solution of ETFS) are over $\Sigma \cup \{\#\}$.

▶ **Example 1.** Let $W = \mathtt{babaaaaabbbab}$, $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $k = 3$, and the set of sensitive substrings be $\{\mathtt{aba}, \mathtt{baa}, \mathtt{aaa}, \mathtt{aab}, \mathtt{bba}\}$. Then $X = \mathtt{babbb\#bab}$ and $X_{\mathrm{ED}} = \mathtt{bab\#aa\#abbb\#bab}$. Note that both $X$ and $X_{\mathrm{ED}}$ satisfy **C1** and **P1**. $X$ is the *shortest* such string and $X_{\mathrm{ED}}$ is a string *closest* to $W$ in terms of edit distance.

In Section 3, we show the following theorem improving the result of [4] by a factor of $|\Sigma|$.

▶ **Theorem 2.** *The ETFS problem can be solved in $\mathcal{O}(kn^2)$ time.*

Our algorithm is based on a non-trivial modification of the classic dynamic programming algorithm for computing the edit distance between two given strings. In particular, the modification is based on the fact that in ETFS we are given a *single* string $W$, and we are asked to *construct* a string $X_{\mathrm{ED}}$ that satisfies **C1** and **P1** and that is closest to $W$. We thus actually fill in the dynamic programming matrix that computes the minimum edit distance between $W$ and a regular expression that is a suitable abstraction of $X_{\mathrm{ED}}$; our algorithm encodes in its recurrence formulae the choices that specify the instance of the regular expression that we eventually output.

In Section 4, we also show that ETFS cannot be solved in strongly subquadratic time unless the Strong Exponential Time Hypothesis (SETH) [11, 10] is false. This is the most technically involved part of the paper.

▶ **Theorem 3.** *The ETFS problem cannot be solved in $\mathcal{O}(n^{2-\delta})$ time, for any $\delta > 0$, unless SETH is false.*

To arrive at this theorem, we reduce the weighted edit distance problem, which is known to admit the same conditional lower bound [2, 7], to the ETFS problem. In particular, given two strings $P$ and $Q$ of length $\Theta(n)$, we construct an instance of ETFS of length $\mathcal{O}(n)$ from the output of which we can infer the insertions corresponding to some optimal alignment of $P$ and $Q$ with respect to the weighted edit distance. Using another suitable instance of ETFS, we can determine the corresponding deletions. That gives us an optimal alignment of $P$ and $Q$, from which we can compute the weighted edit distance of $P$ and $Q$ in $\mathcal{O}(n)$ time.

## 2 Definitions and Notation

Let $T = T[0]T[1]\dots T[n-1]$ be a *string* of length $|T| = n$ over a finite alphabet $\Sigma$ of size $|\Sigma| = \sigma$. By $\Sigma^*$ we denote the set of all strings over $\Sigma$, and by $\Sigma^k$ the set of all length-$k$ strings over $\Sigma$. For two indices $0 \le i \le j \le n-1$, $T[i\,..\,j] = T[i]\dots T[j]$ is the *substring* of $T$ that starts at position $i$ and ends at position $j$ of $T$. By $\varepsilon$ we denote the *empty string* of length 0. A *prefix* of $T$ is a substring of the form $T[0\,..\,j]$, and a *suffix* of $T$ is a substring of the form $T[i\,..\,n-1]$. A prefix (resp. suffix) of a string is *proper* if it is not equal to the string itself. The *reverse* $T[n-1]T[n-2]\dots T[0]$ of $T$ is denoted by $T^R$. Given two strings $U$ and $V$ we say that $U$ has a *suffix-prefix overlap* of length $\ell > 0$ with $V$ if and only if the length-$\ell$ suffix of $U$ is equal to the length-$\ell$ prefix of $V$, i.e., $U[|U| - \ell\,..\,|U| - 1] = V[0\,..\,\ell - 1]$.

We fix a string $W$ of length $n$ over an alphabet $\Sigma$ and an integer $0 < k < n$. We refer to a length-$k$ string or a *pattern* interchangeably. An occurrence of a pattern is uniquely defined by its starting position. Let $\mathcal{S}$ be the set representing the sensitive patterns as positions

over $\{0,\ldots,n-k\}$ with the following closure property: for every $i \in \mathcal{S}$, any $j$ for which $W[j\,..\,j+k-1] = W[i\,..\,i+k-1]$, must also belong to $\mathcal{S}$. That is, if an occurrence of a pattern is in $\mathcal{S}$, then all its occurrences are in $\mathcal{S}$. A substring $W[i\,..\,i+k-1]$ of $W$ is called *sensitive* if and only if $i \in \mathcal{S}$; $\mathcal{S}$ is thus the complete set of occurrences of sensitive patterns. The difference set $\mathcal{I} = \{0,\ldots,n-k\} \setminus \mathcal{S}$ is the set of occurrences of *non-sensitive* patterns.

For any string $U$, we denote by $\mathcal{I}_U$ the set of occurrences in $U$ of non-sensitive length-$k$ strings over $\Sigma$. (We have that $\mathcal{I}_W = \mathcal{I}$.) We call an occurrence $i$ the *t-predecessor* of another occurrence $j$ in $\mathcal{I}_U$ if and only if $i$ is the largest element in $\mathcal{I}_U$ that is less than $j$. This relation induces a *strict total order* on the occurrences in $\mathcal{I}_U$. We call a subset $\mathcal{J}$ of $\mathcal{I}_U$ a *t-chain* if for all elements in $\mathcal{J}$ except the minimum one, their t-predecessor is also in $\mathcal{J}$. For two strings $U$ and $V$, chains $\mathcal{J}_U$ and $\mathcal{J}_V$ are *equivalent*, denoted by $\mathcal{J}_U \equiv \mathcal{J}_V$, if and only if $|\mathcal{J}_U| = |\mathcal{J}_V|$ and $U[u\,..\,u+k-1] = V[v\,..\,v+k-1]$, where $u$ is the $j$-th smallest element of $\mathcal{J}_U$ and $v$ is the $j$-th smallest of $\mathcal{J}_V$, for all $j \le |\mathcal{J}_U|$.

Given two strings $U$ and $V$ the *edit distance* $d_E(U,V)$ is defined as the minimum number of elementary edit operations (letter insertion, deletion, or substitution) that transform one string into the other. Each edit operation can also be associated with a cost: a fixed positive value. Given two strings $U$ and $V$ the *weighted edit distance* $d_{\mathrm{WE}}(U,V)$ is defined as the minimal total cost of a sequence of edit operations to transform one string into the other.

We now formally define ETFS, the main problem considered in this paper.

▶ **Problem 1** (ETFS). *Given a string $W$ of length $n$, an integer $k > 1$, and a set $\mathcal{S}$ (and thus set $\mathcal{I}$), construct a string $X_{ED}$ which is at minimal (weighted) edit distance from $W$ and satisfies the following:*
***C1*** *$X_{ED}$ does not contain any sensitive pattern.*
***P1*** *$\mathcal{I}_W \equiv \mathcal{I}_{X_{ED}}$, i.e., the t-chains $\mathcal{I}_W$ and $\mathcal{I}_{X_{ED}}$ are equivalent.*

We also provide the following auxiliary definitions from [18]. The set of *regular expressions* over $\Sigma$ is defined recursively as follows: (i) $a \in \Sigma \cup \{\varepsilon\}$ is a regular expression. (ii) If $E$ and $F$ are regular expressions, then so are $EF$, $E|F$, and $E^*$, where $EF$ denotes the set of strings obtained by concatenating a string in $E$ and a string in $F$, $E|F$ is the union of the strings in $E$ and $F$, and $E^*$ consists of all strings obtained by concatenating zero or more strings from $E$. Parentheses are used to override the natural precedence of the operators, which places the operator $^*$ highest, the concatenation next, and the operator $|$ last. We say that a string $T$ *matches* a regular expression $E$, if $T$ is equal to one of the strings in $E$.

## 3 ETFS-DP: An $\mathcal{O}(kn^2)$-time Algorithm for ETFS

In this section we describe ETFS-DP, a dynamic programming algorithm that solves ETFS faster than the algorithm proposed in [4]. We describe our algorithm for the unweighted edit distance model for simplicity, but it should be clear that it can be extended to the weighted edit distance model in a straightforward way and with no additional cost. Intuitively, since we are looking for a string $X_{\mathrm{ED}}$ that contains all the non-sensitive patterns of $W$, and in the same order, for each pair $(U,V)$ of non-sensitive patterns of $W$ such that $U$ is the $t$-predecessor of $V$, we can (i) *merge* $U$ and $V$ into $U \cdot V[k-1]$ when $U$ and $V$ have a suffix-prefix overlap of length $k-1$; or (ii) *interleave* $U$ and $V$ constructing a string $UYV$, where $Y$ is a carefully selected string over $\Sigma \cup \{\#\}$, where $\# \notin \Sigma$.

Let us start by defining a regular expression gadget $\oplus$, which encodes all candidate strings that can be used to interleave two non-sensitive patterns while respecting **C1**, and two similar gadgets $\ominus$ and $\otimes$. We will make use of the following regular expression:

$$\Sigma^{<k} = (\underbrace{(a_1|a_2|\ldots|a_{|\Sigma|}|\varepsilon)\ldots(a_1|a_2|\ldots|a_{|\Sigma|}|\varepsilon)}_{k-1 \text{ times}}),$$

where $\Sigma = \{a_1, a_2, \ldots, a_{|\Sigma|}\}$ is the alphabet of $W$. Given a letter $\# \notin \Sigma$, we define

$$\oplus = \#(\Sigma^{<k}\#)^*, \quad \ominus = (\Sigma^{<k}\#)^*, \quad \otimes = (\#\Sigma^{<k})^*.$$

Let $N_0$, $N_1$, …, $N_{|\mathcal{I}|-1}$ be the sequence of non-sensitive patterns as they occur in $W$ from left to right. In [3], $X_{\mathrm{ED}}$ was built by finding an optimal alignment between $W$ and a regular expression $R$ constructed as follows. First, set $R = \ominus N_0$ and then process pairs of non-sensitive patterns $N_i$ and $N_{i+1}$, for all $i \in \{1, \ldots, |\mathcal{I}| - 2\}$: in the $i$-th step, if $N_i$ and $N_{i+1}$ can be merged, append $(N_{i+1}[k-1] \mid \oplus N_{i+1})$ to $R$. Otherwise, append $\oplus N_{i+1}$ to $R$. After processing all pairs, conclude by appending $\otimes$ to $R$. The length of $R$ is in $\mathcal{O}(k|\Sigma|n)$.

The general idea in Algorithm ETFS-DP is to simulate the alignment of $W$ to $R$ without constructing $R$ explicitly. Instead, we use a string $T = \boxminus N_0 \boxplus N_1 \cdots \boxplus N_{|\mathcal{I}|-1} \boxtimes$, where $\boxminus$, $\boxplus$ and $\boxtimes$ are length-1 *placeholders* for $\ominus$, $\oplus$ and $\otimes$, respectively. The length of $T$ is thus only $(k+1)|\mathcal{I}| + 1 = \mathcal{O}(kn)$, leading to an $\mathcal{O}(kn^2)$-time algorithm when aligned to $W$, $|W| = n$.

## 3.1 Dynamic Programming

In a preprocessing phase, we compute a binary array $M$ of length $|\mathcal{I}|$ so that $M[\ell] = 1$ if the $\ell$-th and the $(\ell-1)$-th non-sensitive patterns (in the order given by their occurrences in $W$) can be merged. We set $M[0] = 0$ for completeness. More formally, for all $0 < \ell \leq |\mathcal{I}| - 1$, $M[\ell] = 1$ if $N_{\ell-1}[1\,..\,k-1] = N_\ell[0\,..\,k-2]$, and $M[\ell] = 0$ otherwise.

We then solve ETFS in a dynamic programming fashion by filling in an $(|\mathcal{I}|(k+1)+1) \times (|W|+1)$ matrix $E$. The rows of $E$ correspond to string $T$, and the columns to string $W$. We denote by $E[i][\cdot]$ and $E[\cdot][j]$ the $i$-th row and the $j$-th column of $E$, respectively.

Entry $E[i][j]$, for all $0 \leq i \leq |\mathcal{I}|(k+1)$ and $0 \leq j \leq |W|$, contains the edit distance between (the regular expression corresponding to) $T[0\,..\,i]$ and $W[0\,..\,j-1]$. Rows corresponding to $\boxplus$, i.e., rows with index $i = \ell(k+1)$ for some $\ell \in [1, |\mathcal{I}| - 1]$, implicitly represent a regular expression gadget and must be filled in with ad hoc rules; we will refer to them as *gadgets rows*. In turn, we will name *possibly mergeable* the rows with index $i = \ell(k+1) - 1$ for some $\ell \in [1, |\mathcal{I}| - 1]$, as they must be filled in taking into account the option of merging the corresponding pattern with the preceding one, should it be possible. All other rows of $E$ will be called *ordinary*. In what follows, $I$ is a function such that $I[T[i] \neq W[j-1]] = 1$ if $T[i] \neq W[j-1]$, and 0 otherwise. We give below the recursive formulae that constitute the core of our dynamic programming algorithm.

**Initialization.** Entry $E[0][j]$ contains the edit distance between $\ominus$ and $W[0\,..\,j-1]$ for $j \geq 1$, while $E[0][0] = 0$. Because of the definition of $\ominus$, it is only possible to match up to $k-1$ consecutive letters, after which a mismatch due to # occurs, and hence $E[0][j] = \lceil j/k \rceil$. $E[i][0]$ stores the edit distance between $T[0\,..\,i]$ and the empty prefix $\varepsilon$ of $W$. This distance is minimized by the shortest possible string in each regular expression prefix, obtained by always merging when allowed, and picking the shortest possible string encoded by $\oplus$ when not. This leads to the following formula, where $\ell \in [0, |\mathcal{I}| - 1]$.

$$E[i][0] = \begin{cases} E[i-k-1][0] + 1, & \text{if } i = (\ell+1)(k+1) - 1 \wedge M[\ell] = 1 \text{ (merge)} \\ E[i-1][0] + 1, & \text{otherwise (no merge)} \end{cases} \tag{1}$$

**Ordinary Rows: $i \not\equiv 0 \mod (k+1)$ and $i \not\equiv -1 \mod (k+1)$.** The formula is the same as in the standard algorithm for edit distance [13]: recall that $E[\cdot][j]$ correspond to $W[j-1]$.

$$E[i][j] = \min \begin{cases} E[i-1][j] + 1, & \text{(insert)} \\ E[i][j-1] + 1, & \text{(delete)} \\ E[i-1][j-1] + I[T[i] \neq W[j-1]], & \text{(match or substitute)} \end{cases} \tag{2}$$

**Possibly Mergeable Rows: $i \equiv -1 \mod (k+1)$.** These rows correspond to the last letter of a non-sensitive pattern. The first three options of Equation 3 encode the case where we do not merge, regardless of the value of $M[\ell]$. The last two options, instead, require $M[\ell] = 1$, as a merge does take place. This means that the letters corresponding to the $k$ rows above will not appear in the output string $X_{\mathrm{ED}}$, and thus play no role in the edit distance computation. We thus read the values of row $i-k-1$, corresponding to the last letter of the previous non-sensitive pattern.

$$E[i][j] = \min \begin{cases} E[i-1][j]+1, & \text{(insert)} \\ E[i][j-1]+1, & \text{(delete)} \\ E[i-1][j-1] + I[T[i] \neq W[j-1]], & \text{(match or substitute)} \\ E[i-k-1][j]+1, & \text{if } M[\ell]=1 \text{ (insert and merge)} \\ E[i-k-1][j-1] + I[T[i] \neq W[j-1]], & \text{if } M[\ell]=1 \text{ (match or sub. and merge)} \end{cases} \tag{3}$$

**Gadget Rows: $i \equiv 0 \mod (k+1)$.** A gadget row encodes the possibility of interleaving two non-sensitive patterns with a string that preserves **C1** and **P1** and minimizes the edit distance. Because of the form of the regular expression gadgets, a # can either be inserted or substituted directly after a non-sensitive pattern, or be preceded by another # no more than $k$ positions earlier. This results in the following formula:

$$E[i][j] = \min \begin{cases} E[i-1][j]+1, & \text{(insert)} \\ E[i-1][j-1]+1, & \text{(substitute)} \\ E[i][j-1]+1, \ldots, E[i][\max\{0, j-k\}]+1, & \text{(delete or extend gadget)} \end{cases} \tag{4}$$

The following lemma states that the above formulae correctly compute the edit distance between prefixes of $T$ and prefixes of $W$.

▶ **Lemma 4.** *$E[i][j] = d_E(T[0\,..\,i], W[0\,..\,j-1])$, for all $0 \leq i < |\mathcal{I}|(k+1)$ and $0 < j \leq |W|$, and $E[i][0] = d_E(T[0\,..\,i], \varepsilon)$.*

**Proof.** The correctness of the equations that describe how to fill in entries $E[0][j]$ and $E[i][0]$ follows from the explanation in paragraph "Initialization", and the correctness of Equation 2 follows from the standard dynamic programming algorithm for edit distance [13]. Let us focus on the case of possibly mergeable rows (Equation 3): when merging is not possible, the equation is the same as in the standard algorithm, and therefore it is correct. When merging is possible, we must pick the minimum value among all possible edit operations when we actually choose to merge and among all possible operations when we do not merge, even if we could. The first three rows of Equation 3 correspond to the three possible operations when we do not merge, and are again the same possibilities as the standard algorithm for edit distance; the last two rows correspond to the case where we merge. When we merge, we append the letter corresponding to the possibly mergeable row to the previous non-sensitive pattern. If we were to run the standard algorithm for computing the edit distance between such string and $W$, the row above, where we had to read the values for insertion and match or substitution, would be the one corresponding to the last letter of the previous non-sensitive pattern. These are precisely the values of the last two rows of Equation 3, that are therefore correct.

Consider now the gadget rows. An entry $E[i][j]$ on a gadget row should contain the value of an optimal alignment between $W[0\,..\,j-1]$ and a prefix of $X_{\mathrm{ED}}$ that ends with a #: since # $\notin \Sigma$, it cannot match with any letter of $W$, therefore $I[T[i] \neq W[j-1]] = 1$ always holds. As previously observed, a # can either be inserted or substituted directly after a non-sensitive

pattern, or be preceded by another # no more than $k$ positions earlier. Moreover, it is easy to see that, if an optimal alignment between $W$ and the regular expression $R$ involves a local alignment between $W[i\,..\,j]$ and #$S$# with $|S| = j - i - 1 < k$, then $S = W[i+1\,..\,j-1]$: this is because any alignment with $S \neq W[i+1\,..\,j-1]$ can be improved by replacing $S$ with $W[i+1\,..\,j-1]$. Equation 4 follows from the two observations above: the first two lines compute the cost of appending a # directly after a non-sensitive pattern, that always entails either an insertion or a substitution.

The third row of the equation considers the possibility of interleaving two non-sensitive patterns with a whole string encoded by $\oplus$, or deleting #. ◀

Note that Lemma 4 refers to rows $0 \leq i < |\mathcal{I}|(k+1)$. Let us now look at the last row: even if it was filled in like any other gadget row, since it corresponds to $\otimes$ instead of $\oplus$, its values need to be interpreted in a different way. Namely, the value stored in $E[|\mathcal{I}|(k+1)][j]$, for all $0 \leq j \leq |W|$, is the cost of an optimal alignment between $W[0\,..\,j+e_j-1]$ and a string in $R$ whose length-$(e_j+1)$ suffix is #$W[j\,..\,j+e_j-1]$, where $e_j = \min\{k-1, |W|-j\}$.

Unlike in the standard edit distance algorithm [13], the edit distance between $W$ and any string matching the regular expression $R$ is not necessarily found in its bottom-right entry $E[|\mathcal{I}|(k+1)][|W|]$. Instead, it is found among the rightmost $k$ entries of the last row (in case $X_{\mathrm{ED}}$ ends with a string in $\otimes$), and the rightmost entry of the second-last row (when $X_{\mathrm{ED}}$ ends with the last letter of the last non-sensitive pattern). We thus obtain the following.

▶ **Lemma 5.** *Let $X_{\mathrm{ED}}$ be a solution to ETFS. Then*

$$d_E(X_{\mathrm{ED}}, W) = \min\big\{E\big[|\mathcal{I}|(k+1)-1\big]\big[|W|\big], E\big[|\mathcal{I}|(k+1)\big]\big[|W|\big], E\big[|\mathcal{I}|(k+1)\big]\big[|W|-1\big], \ldots, E\big[|\mathcal{I}|(k+1)\big]\big[|W|-k+1\big]\big\}. \tag{5}$$

## 3.2 Construction of $X_{\mathrm{ED}}$

Once we have computed the edit distance $d$ according to Lemma 5, we need to construct a string $X_{\mathrm{ED}}$ that matches $R$ and is at edit distance $d$ from $W$. To do so, when computing each entry $E[i][j]$ of the matrix for $i, j \geq 1$, we store, in an array $A$, a *pointer* $\langle i', j'\rangle$ to an entry from which the minimum value for $E[i][j]$ was obtained. We then build $X_{\mathrm{ED}}^R$ by following any path from an entry $E[\bar{i}][\bar{j}]$ where the global optimum is stored to $E[0][0]$.

At any step of the construction, let $E[i'][j']$ be the endpoint of the pointer stored for $E[i][j]$ currently considered, i.e., $A[i][j] = \langle i', j'\rangle$. If $\bar{i} = |\mathcal{I}|(k+1)$, i.e., if the minimum is in the last row of $E$, we initialize $X_{\mathrm{ED}}^R$ with $W[\bar{j}\,..\,|W|-1]^R$; otherwise, we just initialize it with the empty string $\varepsilon$. We then enforce the following rules:

**If $\mathbf{i' < i}$,** we append $T[i]$ to $X_{\mathrm{ED}}^R$ when $i$ is not a gadget row and # otherwise. Indeed, the condition is fulfilled when the edge in the path is either diagonal (a match or a substitution in the alignment) or vertical (an insertion in $W$). Moreover, $i'$ can either be equal to $i-1$ or to $i-k-1$ (when we merge two non-sensitive patterns).

**If $\mathbf{i' = i}$ and $\mathbf{i \equiv 0 \mod (k+1)}$,** we append # to $X_{\mathrm{ED}}^R$ followed by $W[j'\,..\,j-2]^R$. Because this happens when we follow a horizontal edge on a gadget row, the solution must include the corresponding substring, that is composed of # and $j - j' - 1$ letters of $W$.

If none of the two cases above happens, we do not write anything, because a horizontal edge in the path corresponds to a deletion in $W$. We denote the above procedure by Algorithm $X_{\mathrm{ED}}$-construct. Lemma 6 guarantees that this construction produces a string that satisfies **C1** and **P1**.

▶ **Lemma 6.** *$X_{\mathrm{ED}}$ returned by Algorithm $X_{\mathrm{ED}}$-construct satisfies **C1** and **P1**.*

**Proof.** Let us start by proving that Algorithm $X_{\mathrm{ED}}$-construct satisfies **C1**. $X^R_{\mathrm{ED}}$ (and thus $X_{\mathrm{ED}}$) is obtained by appending either consecutive letters of $T^R$ (case $i' < i$ for all but gadget rows) or a letter # (all cases for gadget rows) or a number of consecutive letters of $W^R$ (case $i' = i$ for gadget rows and initialization of $X^R_{\mathrm{ED}}$ when the minimum is on the last row of $E$): since $T$ does not contain any sensitive patterns by construction and $\# \notin \Sigma$, we only need to verify that no more than $k-1$ consecutive letters read directly from $W^R$ can ever be appended to $X^R_{\mathrm{ED}}$. Inspect case $i' = i$ for gadget rows: $j - j' - 1$ is the number of entries between entry $E[i][j]$ and the endpoint of the corresponding horizontal pointer $A[i][j]$. The last line of Equation 4 exhibits the only possibilities for a pointer to point a non-adjacent entry on the same row, thus $j' \geq j - k$ and consequently $j - j' - 1 \leq k - 1$. Since both when the path leaves a gadget row and when it goes on on a gadget row a # is appended to $X^R_{\mathrm{ED}}$, no sensitive patterns can be created and therefore $X_{\mathrm{ED}}$ satisfies **C1**.

Let us now show that **P1** is satisfied as well, i.e., $N_0$, $N_1$, $\dots N_{|\mathcal{I}|-1}$ occur in $X_{\mathrm{ED}}$ in the same order as they appear in $W$, and no other length-$k$ string over $\Sigma$ is a substring of $X_{\mathrm{ED}}$. Consider a letter $N_\ell[h] = T[i]$. If $0 \leq h < k-1$, $i$ is an ordinary row. Since any optimal path goes from the entry of $E$ where the minimum is stored to $E[0][0]$, and by construction to leave a row the pointers can only point to an entry in the row directly above the current one (ordinary rows) or in the $(k+1)$-th row above (merge case in the possibly mergeable rows, see Equations 2 and 3), there are only two possibilities: either the path goes through row $i$, i.e., there exists $j$ such that $A[i][j] = \langle i-1, j' \rangle$ is part of the optimal path, or row $i$ is skipped by the path, and thus there exists $j$ such that $A[i+k-h-1][j] = \langle i-h-2, j' \rangle$. Let us observe that in the latter case, all of the rows from $i-h-1$ to $i+k-h-2$ are skipped by the path, while in the first case $A[i+k-h-1][j] = \langle i+k-h-2, j' \rangle$ and no rows are skipped up to $i-h-2$. In the first case, Algorithm $X_{\mathrm{ED}}$-construct will append $N_\ell[h]$ to $X^R_{\mathrm{ED}}$ after $N_\ell[h+1]$ for all $0 \leq h \leq k-1$, then a # right after $N_\ell[0]$, that prevents the making of spurious length-$k$ strings over $\Sigma$ in $X_{\mathrm{ED}}$. In the second case, $N_\ell[h]$ is not explicitly appended to the string: instead, after appending $N_\ell[k-1]$ to $X^R_{\mathrm{ED}}$, the algorithm goes to row $i-h-2$, corresponding to $N_{\ell-1}[k-1]$. Nevertheless, this only happens when the merge condition is satisfied, i.e., when $N_{\ell-1}[1\,..\,k-1] = N_\ell[0\,..\,k-2]$, implying that $N_\ell[h] = N_{\ell-1}[h+1]$ will be appended next to $N_\ell[h+1] = N_{\ell-1}[h+2]$ after $k-h-1$ steps. The order in which $N_0$, $N_1$, $\dots N_{|\mathcal{I}|-1}$ appear in $X_{\mathrm{ED}}$ is by construction the same as they appear in $T$, which in turn is the same as the order they appear in $W$. In no other parts of the algorithm a length-$k$ string over $\Sigma$ is created in $X_{\mathrm{ED}}$. It follows that **P1** is preserved. ◀

## 3.3 Wrapping up

▶ **Lemma 7.** *Algorithm ETFS-DP runs in $\mathcal{O}(kn^2)$ time.*

**Proof.** We first construct string $T$ and array $M$ in $\mathcal{O}(kn)$ time and initialize the first row and the first column of matrix $E$ in $\mathcal{O}(kn)$ time. There are $\mathcal{O}(kn)$ "ordinary", $\mathcal{O}(n)$ "possibly mergeable" and $\mathcal{O}(n)$ "gadget rows", each of size $\mathcal{O}(n)$. Each entry (and its corresponding pointer) on the "ordinary" and "possibly mergeable" rows takes constant time to compute, while the entries (and pointers) on the gadget rows require $\mathcal{O}(k)$ time each. Thus, we can compute all entries and pointers in $\mathcal{O}(kn^2)$ time. Tracing back the pointers and constructing string $X_{\mathrm{ED}}$ takes again $\mathcal{O}(kn)$ time. This results in a total time complexity of $\mathcal{O}(kn^2)$. ◀

Lemmas 4-7 imply Theorem 2.

## 4 A Conditional Lower Bound for ETFS

We prove that, assuming SETH introduced in [11] and [10], ETFS cannot be solved in strongly subquadratic time. We do so by a reduction from the classical edit distance problem, and using the following known conditional lower bound for it: for all $\delta > 0$, the edit distance $d_E$ between two strings of length $\Omega(n)$ cannot be computed in $\mathcal{O}(n^{2-\delta})$ time without violating SETH [2], and hence the well-known quadratic-time solution of [13] for computing the edit distance between two strings of length $\mathcal{O}(n)$ is optimal up to subpolynomial factors. Bringmann and Künnemann [7] proved that this is also true for weighted edit distance, where each operation (insertion, deletion, substitution and match) has a corresponding fixed non-negative cost (respectively $c_\mathrm{i}, c_\mathrm{d}, c_\mathrm{s}, c_\mathrm{m}$), and the following conditions, which we will call the BK conditions, hold: (i) $c_\mathrm{i} + c_\mathrm{d} > c_\mathrm{m}$, (ii) $c_\mathrm{i} + c_\mathrm{d} > c_\mathrm{s}$, and (iii) $c_\mathrm{m} \neq c_\mathrm{s}$.

Let $P$ and $Q$ be two arbitrary strings over $\Sigma$, both of length $\Theta(n)$, and without loss of generality $1 \leq |P| \leq |Q|$. We would like to compute the weighted edit distance between $P$ and $Q$ with the following associated costs: $c_\mathrm{i} = 2.5, c_\mathrm{d} = 2.5, c_\mathrm{s} = 1$, $c_\mathrm{m} = 0$. These costs satisfy the BK conditions. Let $c = (c_\mathrm{i}, c_\mathrm{d}, c_\mathrm{s}, c_\mathrm{m})$ and $d_c$ be the weighted edit distance with associated costs $c$. Assuming SETH is true, there is no algorithm for computing $d_{(2.5,2.5,1,0)}(P, Q)$ in $\mathcal{O}(n^{2-\delta})$ time, for any $\delta > 0$ [7]. In order to prove that ETFS cannot be solved in strongly subquadratic time either, we will compute $d_{(2.5,2.5,1,0)}(P, Q)$, by solving two instances of ETFS on a string of length $\mathcal{O}(n)$ and using an additional $\mathcal{O}(n)$ number of operations. Thus if ETFS is solvable in $\mathcal{O}(n^{2-\delta})$ time, for any $\delta > 0$, SETH is false.

Let us now show the first instance of the ETFS. We define a new alphabet $\Sigma' = \Sigma \sqcup \{\mathtt{a}, \mathtt{b}, \mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$ and a new string $U(P, Q) = F_1F_2F_3F_4$ over $\Sigma'$ as follows:

$$F_1 = (\mathtt{aab})^{2x+1}\mathtt{aae}, \quad F_2 = \prod_{i=0}^{|P|-1} \mathtt{c}_1\mathtt{d}P[i]\mathtt{c}_2\mathtt{c}_3, \quad F_3 = (\mathtt{aae})^{2x-1}\mathtt{aa}, \quad F_4 = \prod_{i=0}^{|Q|-1} \mathtt{c}_1\mathtt{f}Q[i]\mathtt{c}_2\mathtt{c}_3$$

where $x = 2|Q|$, and the product denotes the concatenation operation on strings. We also set $k = 5$ and define the set $\mathcal{I}$ of non-sensitive pattern occurrences over $U$ as follows:

$$\mathcal{I} \;=\; \{0, 3, 6, 9, \ldots, 6x\} \;\cup\; \{6x + 6, 6x + 11, 6x + 16, \ldots, 6x + 1 + 5|P|\}.$$

In particular, $U(P, Q)$ is the string input to the first instance of ETFS. The construction above gives us the following sequence of *non-sensitive* patterns:

$$\begin{aligned} &\mathtt{aabaa}, \mathtt{aabaa}, \mathtt{aabaa}, \ldots, \mathtt{aabaa} && (2x + 1 \text{ occurrences}) \\ &\mathtt{c}_1\mathtt{d}P[0]\mathtt{c}_2\mathtt{c}_3, \mathtt{c}_1\mathtt{d}P[1]\mathtt{c}_2\mathtt{c}_3, \mathtt{c}_1\mathtt{d}P[2]\mathtt{c}_2\mathtt{c}_3, \ldots, \mathtt{c}_1\mathtt{d}P[|P| - 1]\mathtt{c}_2\mathtt{c}_3 && (|P| \text{ occurrences}). \end{aligned}$$

It is easy to verify that the set $\mathcal{I}$ of occurrences of non-sensitive patterns (and thus the complementary set $\mathcal{S}$) has the closure property requested by ETFS. The resulting regular expression $R$ is

$$R = \ominus\, \mathtt{aabaa} \oplus \mathtt{aabaa} \oplus \ldots \oplus \mathtt{aabaa} \oplus \mathtt{c}_1\mathtt{d}P[0]\mathtt{c}_2\mathtt{c}_3 \oplus \mathtt{c}_1\mathtt{d}P[1]\mathtt{c}_2\mathtt{c}_3 \oplus \ldots \oplus \mathtt{c}_1\mathtt{d}P[|P|-1]\mathtt{c}_2\mathtt{c}_3 \;\otimes.$$

We will prove that it is optimal to align the first $x + 1$ patterns with $F_1$, a gadget $\oplus$ with $F_2$, the next $x$ patterns with $F_3$ and the final $|P|$ patterns with $F_4$. Then, we will show that the alignment of those last patterns with $F_4$ corresponds to an alignment of $P$ and $Q$.

We call the occurrences of $\mathtt{aabaa}$ and $\mathtt{c}_1\mathtt{d}P[i]\mathtt{c}_2\mathtt{c}_3$ in the regular expression $R$, or in any string in the regular language corresponding to $R$, *AB-patterns* and *P-patterns*, respectively. Notice that these non-sensitive patterns are substrings of $F_1$ and $F_2$ and that we cannot merge any two consecutive non-sensitive patterns.

Recall that the output $X_{\mathrm{ED}}$ of ETFS is a string with minimal edit distance to $U$ in that language. One alignment of $U$ and $R$, which we denote by $\mathcal{A}_{U/R}$ and that we will later show to be optimal under unit cost for insertion, deletion and substitution and zero cost for match, is as follows:

- We align $F_1$ with the first $x+1$ AB-patterns interleaved by `#`'s as illustrated below. The cost of this alignment is $x+1$ substitutions.

```
aabaabaabaabaabaabaabaab...aabaae
aabaa#aabaa#aabaa#aabaa#...aabaa#
```

- We align $F_2$ with a single gadget $\oplus$ suitably expanded as shown below. The cost of this alignment is $|P|$ substitutions. Recall that we have to use a `#` after every $k-1=4$ letters, so as not to introduce any new length-$k$ substrings that would violate property **P1**.

```
c1dP[0]c2c3c1dP[1]c2c3c1dP[2]c2c3c1dP[3]c2c3...c1dP[|P| − 1]c2c3
c1dP[0]c2# c1dP[1]c2# c1dP[2]c2# c1dP[3]c2# ...c1dP[|P| − 1]c2#
```

- We align $F_3$ with the remaining $x$ AB-patterns interleaved by `#`'s as illustrated below. The cost of this alignment is $2x-1$ substitutions.

```
aaeaaeaaeaaeaaeaaeaaeaae...aaeaa
aabaa#aabaa#aabaa#aabaa#...aabaa
```

- We align $F_4$ with the final $|P|$ P-pattern occurrences according to an optimal alignment $\mathcal{A}_{P/Q}$ of $P$ and $Q$ with respect to cost $c$. Let `p` and `q` denote placeholders for letters of $P$ and $Q$, respectively. For each edit operation in $\mathcal{A}_{P/Q}$ (insertion of `q`, deletion of `p`, substitution or match between `p` and `q`), we align in $\mathcal{A}_{U/R}$ the corresponding fragment of $F_4$ and the P-pattern of $R$ as follows.

| **Insertion** | **Deletion** | **Substitution or Match** |
|---|---|---|
| c1f q c2c3 | - - - - - - | - c1f q c2c3 |
| # f q c2c3 | # c1d p c2c3 | # c1d p c2c3 |

  When inserting a letter of $Q$, rather than paying 5 consecutive gaps opposite to fragment `c1fqc2c3` of $F_4$, we extend the gadget $\oplus$ of $R$ with `#fqc2c3`, to pay only one (unavoidable) substitution for `#`. Deleting a letter of $P$, instead, results in 6 gaps in $\mathcal{A}_{U/R}$. Finally, substitutions and matches in $\mathcal{A}_{P/Q}$ result in the same alignment in $\mathcal{A}_{U/R}$, with the cost being, respectively, 3 and 2 according to whether $\mathtt{q}=\mathtt{p}$ or not. Therefore, it turns out that the cost of this last fragment of alignment $\mathcal{A}_{U/R}$ equals $d_{(1,6,3,2)}(P,Q)$.

We next show that it is possible to express $d_{(1,6,3,2)}(P,Q)$ in terms of $d_{(2.5,2.5,1,0)}(P,Q)$, because symmetry will greatly simplify things later on, when we swap $P$ and $Q$.

▶ **Lemma 8.** *Let $c$ and $c'$ be two costs. We write $c \sim c'$ if for any alphabet $\Sigma$ and for all $P,Q \in \Sigma^*$, the set of optimal alignments of $P$ and $Q$ with respect to cost $c$ is equal to the set of optimal alignments of $P$ and $Q$ with respect to cost $c'$. Then*

1. *$c \sim \alpha c$ for all $\alpha \in \mathbb{R}_{>0}$.*
2. *$c \sim (c_i+\alpha, c_d, c_s+\alpha, c_m+\alpha)$ for all $\alpha \in \mathbb{R}$.*
3. *$c \sim (c_i, c_d+\alpha, c_s+\alpha, c_m+\alpha)$ for all $\alpha \in \mathbb{R}$.*

**Proof.** Let the number of insertions, deletions, substitutions and matches in some alignment of $P$ and $Q$ be $n_i$, $n_d$, $n_s$ and $n_m$ respectively. We know that $n_i+n_s+n_m=|Q|$ and $n_d+n_s+n_m=|P|$. So the transformations 1, 2, and 3 of $c$ given in the lemma statement change the costs of alignments from $d$ to $\alpha d$, $d+\alpha|Q|$ and $d+\alpha|P|$ respectively. The costs of alignments are all strictly increasing in $d$, so the optimal alignments are preserved. ◀

By applying transformation 2 of Lemma 8 with $\alpha = 1.5$ and then transformation 3 of Lemma 8 with $\alpha = -3.5$, we obtain

$$d_{(1,6,3,2)}(P,Q) = d_{(2.5,2.5,1,0)}(P,Q) - 1.5|Q| + 3.5|P|. \tag{6}$$

By summing up the costs of the alignment $\mathcal{A}_{U/R}$ detailed above and using Equation 6, we get

$$d_E(U, X_{\mathrm{ED}}) \leq 4.5(|P| + |Q|) + d_{(2.5,2.5,1,0)}(P,Q), \tag{7}$$

which we can bound by $3|P| + 7|Q|$, because $d_{(2.5,2.5,1,0)}(P,Q) \leq 2.5(|Q| - |P|) + |P|$, corresponding to the cost of deleting the $(|Q| - |P|)$ extra letters of $Q$ (recall that $|P| \leq |Q|$) and substituting the remaining $|P|$ letters. In Lemma 9 we prove that alignment $\mathcal{A}_{U/R}$ is indeed optimal and equality holds in Equation 7.

▶ **Lemma 9.** *Alignment $\mathcal{A}_{U/R}$ is optimal. Moreover, from any output $X_{ED}$ of ETFS on $U$ we can obtain a supersequence $P'$ of $P$ in $\mathcal{O}(|Q|)$ time such that $d_c(P,Q) = |P'| - |P| + d_c(P',Q)$ and there exists an optimal alignment of $P'$ and $Q$, which does not use any insertions.*

The reader can probably share the intuition that alignment $\mathcal{A}_{U/R}$ is optimal, at least for the part $F_1F_2F_3$ of string $U$. We prove that indeed no AB-pattern is aligned to any part of $F_4$ and that no $P$-pattern is aligned to $F_1F_2F_3$ (see Example 10). The proof of Lemma 9 consists of a case analysis combined with basic counting and bounding arguments, for which we refer the reader to the full version of this paper.

▶ **Example 10.** Let $P = \mathtt{KITTEN}$ and $Q = \mathtt{SITTING}$ over $\Sigma = \{\mathtt{E}, \mathtt{G}, \mathtt{I}, \mathtt{K}, \mathtt{N}, \mathtt{S}, \mathtt{T}\}$. We define a new alphabet $\Sigma' = \Sigma \sqcup \{\mathtt{a}, \mathtt{b}, \mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$ and a new string $U(P,Q) = F_1F_2F_3F_4$ over $\Sigma'$ as follows (recall that $x = 2|Q|$, so $2x + 2 = 4Q + 2 = 30$):

$$\begin{aligned} F_1 &= \mathtt{aabaabaabaabaabaabaabaab} \ldots \mathtt{aabaae} \\ F_2 &= \mathtt{c_1dKc_2c_3c_1dIc_2c_3c_1dTc_2c_3c_1dTc_2c_3c_1dEc_2c_3c_1dNc_2c_3} \\ F_3 &= \mathtt{aaeaaeaaeaaeaaeaaeaaeaae} \ldots \mathtt{aaeaa} \\ F_4 &= \mathtt{c_1fSc_2c_3c_1fIc_2c_3c_1fTc_2c_3c_1fTc_2c_3c_1fIc_2c_3c_1fNc_2c_3c_1fGc_2c_3} \end{aligned}$$

We also set $k = 5$ and define the set $\mathcal{I}$ of non-sensitive pattern occurrences over $U$ as follows:

$$\mathcal{I} \;=\; \{0, 3, 6, 9, \ldots, 6x\} \;\cup\; \{6x + 6, 6x + 11, 6x + 16, \ldots, 6x + 1 + 5|P|\}.$$

We thus have the following sequence of occurrences of non-sensitive patterns:

$$\begin{aligned} &\mathtt{aabaa}, \mathtt{aabaa}, \mathtt{aabaa}, \ldots, \mathtt{aabaa} && \text{(29 occurrences)} \\ &\mathtt{c_1dKc_2c_3}, \mathtt{c_1dIc_2c_3}, \mathtt{c_1dTc_2c_3c_1dTc_2c_3}, \mathtt{c_1dEc_2c_3}, \mathtt{c_1dNc_2c_3} && \text{(6 occurrences).} \end{aligned}$$

Therefore, the corresponding regular expression $R$ is

$$R = \ominus\, \mathtt{aabaa} \oplus \ldots \oplus \mathtt{aabaa} \oplus \mathtt{c_1dKc_2c_3} \oplus \mathtt{c_1dIc_2c_3} \oplus \mathtt{c_1dTc_2c_3c_1dTc_2c_3} \oplus \mathtt{c_1dEc_2c_3} \oplus \mathtt{c_1dNc_2c_3} \otimes.$$

We now show the crucial fragment of alignment $\mathcal{A}_{U/R}$: how $F_4$ is aligned with the $P$-patterns.

```
-c1fSc2c3-c1fIc2c3-c1fTc2c3-c1fTc2c3-c1fIc2c3-c1fNc2c3c1fGc2c3
#c1dKc2c3#c1dIc2c3#c1dTc2c3#c1dTc2c3#c1dEc2c3#c1dNc2c3# fGc2c3
```

Observe that the cost of the above alignment under unit cost equals to 15: the cost of 4 $P$-pattern matches (8), plus the cost of 2 $P$-pattern substitutions (6), plus the cost of 1 gadget insertion (1). It can be readily verified that $d_{(1,6,3,2)}(\mathtt{KITTEN}, \mathtt{SITTING}) = 15$.

Lemma 9 implies the following result.

▶ **Corollary 11.** $d_E(U, X_{\mathrm{ED}}) = 4.5(|P| + |Q|) + d_{(2.5,2.5,1,0)}(P, Q)$.

Given that constructing $U$ takes $\mathcal{O}(n)$ time, Corollary 11 tells us that, if we can compute $d_E(U, X_{\mathrm{ED}})$ in strongly subquadratic time, then we can also compute $d_c$ between any two strings in strongly subquadratic time contradicting SETH. In fact, to prove that the *output string* of ETFS cannot be computed in strongly subquadratic time either, we show that $d_{(2.5,2.5,1,0)}$ can be obtained by solving ETFS twice and $\mathcal{O}(n)$ additional operations.

By Lemma 9, from the output $X_{\mathrm{ED}}$ of the ETFS algorithm, we can obtain a supersequence $P'$ of $P$ in $\mathcal{O}(n)$ time such that $d_c(P, Q) = d_c(P, P') + d_c(P', Q)$ and no insertions are required to optimally align $P'$ and $Q$. There also exists a supersequence $Q'$ of $Q$ such that $d_c(P, Q) = d_c(P, P') + d_c(Q', Q) + d_c(P', Q')$ and some optimal alignment of $P'$ and $Q'$ which aligns each $P'[i]$ with $Q'[i]$ through either a match or a substitution. One such $Q'$ is the string obtained by taking the alignment of $P'$ and $Q$ given by ETFS and inserting aligned letters of $P'$ into the gaps of $Q$. The edit distance of $Q$ and $P$ is

$$d_c(P, P') + d_c(Q, Q') + d(P', Q') = |P'| - |P| + |Q'| - |Q| + \sum_{i=0}^{|P'|-1} I[P'[i] \neq Q'[i]], \qquad (8)$$

which can be computed in $\mathcal{O}(n)$ time once we know $P'$ and $Q'$.

Note that by using ETFS on $U(Q, P')$, we could find a supersequence $Q''$ of $Q$ such that $d_c(P, Q) = d_c(P, P') + d_c(Q, Q'') + d_c(P', Q'')$ and no deletions are required to optimally align $P'$ and $Q''$. It is not necessarily the case that we do not need any more insertions, though, as optimal alignments are not unique. We now show that we can still compute an appropriate $Q'$ by changing $c$.

Let $\mathcal{Q}_c$ be the set of supersequences $Q''$ of $Q$ with minimal $d_c(Q'', Q) + d_c(P', Q'')$ and no deletions needed in the alignment of $P'$ and $Q''$. Note that there exists some $Q' \in \mathcal{Q}_c$ such that $|P'| = |Q'|$. Increasing the cost of deletion by $\epsilon$, $d_c(Q'', Q) + d_c(P', Q'')$ increases by at least $\epsilon(|P'| - |Q|)$ with equality if and only if $|Q''| = |P'|$. Since $|Q'| = |P'|$, no deletions implies no insertions. Therefore it suffices to find the insertions, when aligning $P'$ and $Q$ with weights $c' = (2.5, 2.5 + \epsilon, 1, 0)$ for some $\epsilon > 0$. We find these insertions by running the ETFS algorithm on $U(G(Q), G(P'))$ with $k = 5$, where $G(V) = \prod_{i=0}^{|V|-1}(V[i]\mathtt{g})$ for any string $V \in (\Sigma \sqcup \{\mathtt{a}, \mathtt{b}, \mathtt{c}_1, \mathtt{c}_2, \mathtt{c}_3, \mathtt{d}, \mathtt{e}, \mathtt{f}\})^*$, and with the set of non-sensitive pattern occurrences

$$\mathcal{I} \;=\; \{0, 3, 6, 9, \ldots, 6x'\} \;\cup\; \{6x' + 6, 6x' + 11, 6x' + 16, \ldots, 6x' + 1 + |Q|\},$$

where $x' = 2|G(P')|$. The solution to this new problem corresponds to an optimal alignment of $G(Q)$ and $G(P')$ with $c = (2.5, 2.5, 1, 0)$, which in its turn corresponds to an optimal alignment of $Q$ and $P'$ with weights $c' = (5, 5, 1, 0) \sim (2.5, 2.5 + 5, 1, 0)$ by Lemma 8. We first carefully define what properties such a corresponding alignment should satisfy, and then prove that all optimal alignments of $G(Q)$ and $G(P')$ are indeed of this form.

▶ **Definition 12.** *The alignment of $G(P')$ and $G(Q)$ corresponding to an alignment $\mathcal{A}_{P'/Q}$ of $P'$ and $Q$ is defined as follows:*

- *If $P'[i]$ is aligned with $Q[i]$ in $\mathcal{A}_{P'/Q}$, then $G(P')[2i]$ and $G(P')[2i + 1]$ are aligned with $G(Q)[2i]$ and $G(Q)[2i + 1]$, respectively.*
- *If $P'[i]$ is deleted in $\mathcal{A}_{P'/Q}$, then $G(P')[2i]$ and $G(P')[2i + 1]$ are deleted.*
- *If $Q[i]$ is inserted in $\mathcal{A}_{P'/Q}$, then $G(Q)[2i]$ and $G(Q)[2i + 1]$ are inserted.*

▶ **Lemma 13.** *Let $P', Q \in \Sigma^*$ such that there exits an optimal alignment of $P'$ and $Q$ which does not include any insertions. Each optimal alignment of $G(P')$ and $G(Q)$ with respect to cost $c = (2.5, 2.5, 1, 0)$ corresponds to an optimal alignment of $P'$ and $Q$ with weights $c' = (5, 5, 1, 0)$.*

**Proof.** Let the number of insertions, deletions, substitutions and matches in some optimal alignment $\mathcal{A}_{P'/Q}$ of $P'$ and $Q$ be $w$, $x$, $y$ and $z$ respectively. The cost of $\mathcal{A}_{P'/Q}$ with respect to $c'$ is $5w + 5x + y$. The corresponding alignment of $G(P')$ and $G(Q)$ has $2w$ insertions, $2x$ deletions, $y$ substitutions and $2z + y$ matches, and its cost with respect to $c$ is $(2w)\cdot 2.5 + (2x)\cdot 2.5 + y\cdot 1 + (2z+y)\cdot 0 = 5w + 5x + y$. Therefore $d_c(G(P'), G(Q)) \leq d_{c'}(P', Q)$. It remains to be shown that equality holds.

Consider an optimal alignment $\mathcal{A}_{G(P')/G(Q)}$ of $G(P')$ and $G(Q)$. We will show that we can transform $\mathcal{A}_{G(P')/G(Q)}$ into one corresponding to an alignment of $P'$ and $Q$ without increasing the edit distance. Consider the rightmost $P'[i]$ and $Q[j]$ where the corresponding alignment fails, and call them $x$ and $y$. There are 13 possibilities for their alignment:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| --xg | -xg | -x-g | -xg | -xg- | x-g | xg | xg- | x--g | x-g | x-g- | xg- | xg-- |
| ygxg | ygg | yxgg | yxg | yxgg | ygg | yg | ygg | .ygg | .yg | .ygg | .yg | ..yg |

Blue letters are original letters of $G(Q)$, red letters are deleted letters from $G(P')$, dots are arbitrary strings and dashes denote gaps. Note that configurations 1, 7 and 13 are already properly aligned. Moreover, the cost can be reduced for configurations 2, 3, 4, 5, 6, 8, 9, 11 and 12 by deleting red letters and shifting blue ones. This only leaves configuration 10. Here there are 3 subcases:

- If `x` is aligned with an `x`, there must be a `g` between `x` and `y`. We can align this `g` with $G(P')[2i]$ and move to configuration 13 without increasing the cost nor changing letters.
- If `x` is aligned with an `x`, we move an adjacent inserted letter to this place and reduce the cost, which is a contradiction.
- Otherwise, `x` is aligned with a different letter. In this case we can realign it with `y` without increasing the cost or changing letters.

Since there is a corresponding alignment for the output string, equality holds. ◀

Therefore the output string is equal to $G(Q')$ for some $Q' \in \mathcal{Q}_c$. We can infer $Q'$ in $\mathcal{O}(n)$ time and compute $d_c(P, Q)$ using Equation 8. However, since $d_c(P, Q)$ could not be computed in strongly subquadratic time given SETH, we conclude that ETFS cannot be computed in strongly subquadratic time either, unless SETH is false, thus proving Theorem 3.

## 5 Final Remarks

The following questions remain unanswered. Can ETFS be solved in $\mathcal{O}(n^2)$ time? Can ETFS be solved in strongly subquadratic time when $|\mathcal{S}| = \mathcal{O}(1)$?

## References


1 O. Abul, F. Bonchi, and F. Giannotti. Hiding sequential and spatiotemporal patterns. *IEEE Transactions on Knowledge and Data Engineering*, 22(12):1709–1723, 2010.

2 A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *47th ACM Annual Symposium on Theory of Computing (STOC)*, pages 51–58, 2015.

3 G. Bernardini, H. Chen, A. Conte, R. Grossi, G. Loukides, N. Pisanti, S. Pissis, and G. Rosone. String sanitization: A combinatorial approach. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 627–644, 2019.

4 G. Bernardini, H. Chen, A. Conte, R. Grossi, G. Loukides, N. Pisanti, S. Pissis, G. Rosone, and M. Sweering. Combinatorial algorithms for string sanitization. *arXiv*, 2019.

5 G. Bernardini, H. Chen, G. Fici, G. Loukides, and S. P. Pissis. Reverse-safe data structures for text indexing. In *Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 199–213, 2020.

6 L. Bonomi, L. Fan, and H. Jin. An information-theoretic approach to individual sequential data sanitization. In *9th ACM International Conference on Web Search and Data Mining (WSDM)*, pages 337–346, 2016.

7 K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *56th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 79–97, 2015.

8 U.S. Department of Health & Human Services. Health Insurance Portablility and Accountability Act. `https://aspe.hhs.gov/report/health-insurance-portability-and-accountability-act-1996`, 1996.

9 R. Gwadera, A. Gkoulalas-Divanis, and G. Loukides. Permutation-based sequential pattern hiding. In *13th IEEE International Conference on Data Mining (ICDM)*, pages 241–250, 2013.

10 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and Systems Sciences*, 62(2):367–375, 2001.

11 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and Systems Sciences*, 63(4):512–530, 2001.

12 L. Jin, C. Li, and R. Vernica. SEPIA: estimating selectivities of approximate string predicates in large databases. *The VLDB Journal*, 17(5):1213–1229, 2008.

13 V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.

14 A. Liu, K. Zhengy, L. Liz, G. Liu, L. Zhao, and X. Zhou. Efficient secure similarity computation on encrypted trajectory data. In *31st IEEE International Conference on Data Engineering (ICDE)*, pages 66–77, 2015.

15 G. Loukides and R. Gwadera. Optimal event sequence sanitization. In *SIAM International Conference on Data Mining (SDM)*, pages 775–783, 2015.

16 W. Lu, X. Du, M. Hadjieleftheriou, and B. C. Ooi. Efficiently supporting edit distance based string similarity search using $B^+$-trees. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):2983–2996, 2014.

17 B. Malin and L. Sweeney. Determining the identifiability of DNA database entries. In *American Medical Informatics Association Annual Symposium (AMIA)*, pages 537–541, 2000.

18 E. W. Myers and W. Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.

19 European Parliament. General Data Protection Regulation. `http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf`.

20 G. Poulis, S. Skiadopoulos, G. Loukides, and A. Gkoulalas-Divanis. Apriori-based algorithms for km-anonymizing trajectory data. *Transactions on Data Privacy*, 7:165–194, 2014.

21 J. Shang, J. Peng, and J. Han. MACFP: Maximal Approximate Consecutive Frequent Pattern Mining under edit distance. In *SIAM International Conference on Data Mining (SDM)*, pages 558–566, 2016.

22 H. J. Smith, T. Dinev, and H. Xu. Information privacy research: An interdisciplinary review. *MIS Quarterly*, 35(4):989–1015, 2011.

23 M. Terrovitis, G. Poulis, N. Mamoulis, and S. Skiadopoulos. Local suppression and splitting techniques for privacy preserving publication of trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1466–1479, 2017.

24 Z. Wen, D. Deng, R. Zhang, and R. Kotagiri. 2ED: An Efficient Entity Extraction Algorithm using two-level Edit-Distance. In *35th IEEE International Conference on Data Engineering (ICDE)*, pages 998–1009, 2019.