# Hide and Mine in Strings: Hardness and Algorithms

Giulia Bernardini[1], Alessio Conte[2], Garance Gourdel[3], Roberto Grossi[2,4], Grigorios Loukides[5]
Nadia Pisanti[2,4], Solon P. Pissis[4,6,7], Giulia Punzi[2], Leen Stougie[4,6,7], Michelle Sweering[6]
[1]University of Milano - Bicocca, Italy
[2]Università di Pisa, Italy
[3]Inria Rennes, École normale supérieure, ENS Paris-Saclay, France
[4]ERABLE Team, France
[5]King's College London, United Kingdom
[6]CWI, The Netherlands
[7]Vrije Universiteit, The Netherlands
[1]giulia.bernardini@unimib.it, [2]{alessio.conte,roberto.grossi,nadia.pisanti}@unipi.it, giulia.punzi@phd.unipi.it
[3]garance.gourdel@ens-paris-saclay.fr, [5]grigorios.loukides@kcl.ac.uk, [6]{solon.pissis,leen.stougie,michelle.sweering}@cwi.nl

***Abstract*—We initiate a study on the fundamental relation between data sanitization (i.e., the process of hiding confidential information in a given dataset) and frequent pattern mining, in the context of sequential (string) data. Current methods for string sanitization hide confidential patterns introducing, however, a number of spurious patterns that may harm the utility of frequent pattern mining. The main computational problem is to minimize this harm. Our contribution here is twofold. First, we present several hardness results, for different variants of this problem, essentially showing that these variants cannot be solved or even be approximated in polynomial time. Second, we propose integer linear programming formulations for these variants and algorithms to solve them, which work in polynomial time under certain realistic assumptions on the problem parameters.**

***Index Terms*—data privacy, data sanitization, knowledge hiding, frequent pattern mining, string algorithms**

## I. Introduction

A string is a sequence of letters over some alphabet $\Sigma$. Strings are commonly used to represent individuals' data in domains ranging from transportation to web analytics and bioinformatics. For example, a string can represent a user's location profile, with each letter corresponding to a visited location [28], a user's purchasing history, with each letter corresponding to a purchased product [2], or a patient's genome sequence, with each letter corresponding to a DNA base [20]. Mining patterns from such strings is thus useful in a gamut of applications, including route planning [8], marketing [2], and clinical diagnostics [20]. To support these applications while preserving privacy, strings representing individuals' data are often being disseminated after sanitization [1], [27].

In this paper, we study the fundamental relation between *data sanitization* [1], [4], [27] (also known as *knowledge hiding*) and *frequent pattern mining* [19], [22], [25]. The objective of frequent pattern mining in strings is to obtain all patterns occurring frequently enough in a string, or in a collection of strings. There may also be constraints for the mined strings (e.g., to be of fixed length $k$ [3], [9]). In string sanitization, the privacy objective is to transform a string to ensure that a given set of *sensitive patterns*, modeling confidential knowledge, does not occur in the sanitized version of the string; sensitive patterns are selected based on domain expertise [4], [15], [27]. This transformation may incur some utility loss that should be minimized. Recent methods achieve this using combinatorial algorithms [4], [5]. Let $W$ be the input string over $\Sigma$, $k > 0$ be an integer, and $\mathcal{S}$ be the set of sensitive length-$k$ substrings. These methods construct a string $X$ such that: (I) $X$ contains no element of $\mathcal{S}$ as a substring; (II) the total order and thus the frequency of all non-sensitive length-$k$ substrings of $W$ is preserved in $X$; and (III) the length of $X$ is minimized [4], or the edit distance between $W$ and $X$ is minimized [5]. These methods work by copying carefully selected substrings of $W$ into $X$ and separating them by a special letter $\# \notin \Sigma$.

**Example 1.** Let $W = \texttt{GACAAAAACCCAT}$, $k = 3$, and the set of sensitive patterns $\mathcal{S} = \{\texttt{ACA}, \texttt{CAA}, \texttt{AAA}, \texttt{AAC}, \texttt{CCA}\}$. Further, let $X_{\mathrm{TR}} = \texttt{GAC\#ACC\#CCC\#CAT}$, $X_{\mathrm{MIN}} = \texttt{GACCC\#CAT}$ and $X_{\mathrm{ED}} = \texttt{GAC\#AA\#ACCC\#CAT}$ be three sanitized strings. All three strings contain *no sensitive pattern* and preserve the *total order* and thus the *frequency* of all non-sensitive length-3 patterns of $W$: $X_{\mathrm{TR}}$ is the trivial solution of interleaving the non-sensitive length-3 patterns of $W$ with $\#$; $X_{\mathrm{MIN}}$ is the *shortest* possible such string [4]; and $X_{\mathrm{ED}}$ is a string *closest* to $W$ in terms of edit distance [5].

Unfortunately, as noted in [4], the occurrences of $\#$ reveal the *locations* of sensitive patterns and thus must be ultimately replaced by letters of the original alphabet $\Sigma$. This replacement gives rise to another string over $\Sigma$, which we denote by $Z$. However, this replacement may create spurious patterns that could not be mined from $X$ at a minimum frequency threshold $\tau$ but would be mined from $Z$ at the same frequency threshold. These patterns are referred to as $\tau$-*ghosts*.

We investigate the crucial interplay between $\#$ replacements and $\tau$-ghosts, posing here the following question that, to the best of our knowledge, has not been addressed: *Given a string $X$ containing $\#$'s, a positive integer $k$, and a positive integer $\tau$, how should we replace the $\#$'s in $X$ with letters in $\Sigma$, so that the number of length-$k$ $\tau$-ghosts in the resulting string $Z$ is minimized?* This question helps preserving the

accuracy of frequent pattern mining and tasks based on it (e.g., pattern-based clustering [17] and classification [24], as well as sequential rule mining [26]) that we may not know a priori.

The above question is also of quite general interest, as it applies to sequential datasets that may have occurrences of a special letter for a variety of reasons beyond data sanitization. This special letter, denoted here by # for consistency, represents some information that is *missing* from these datasets. For instance, in genome sequencing data, # corresponds to an unknown DNA base [18]; in databases, # represents a value that has not been recorded [7], [12]; and in masked data outputted by other privacy-preserving methods [6], # is introduced deliberately to achieve their privacy goal.

Like in data outputted by sanitization methods, the occurrences of # in other string datasets often have to be replaced. For example, since the DNA alphabet consists of four letters (A, C, G, and T), off-the-shelf algorithms for processing DNA data use a two-bits-per-base encoding to represent the DNA alphabet. In order to use these algorithms with input strings containing unknown bases, we would have to amend them to work on the extended alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \#\}$. This solution may have a negative impact on the time efficiency of the algorithms or the space efficiency of the data structures they use. Thus, instead, in several state-of-the-art DNA data processing tools (e.g., [21]), the occurrences of # are replaced by an arbitrarily chosen letter from the DNA alphabet, so that off-the-shelf algorithms can be directly employed. This, however, may introduce a large number of spurious patterns, negatively affecting the accuracy of DNA analyses.

Replacing the occurrences of # in a database is often needed to be able to perform frequent pattern mining with off-the-shelf algorithms [12]. To this end, the occurrences of # are commonly replaced by some statistical estimate, such as the most frequent value [12], [16]. However, such a replacement does not generally maintain the accuracy of frequent pattern mining, since it may introduce many spurious patterns [12].

**Example 2.** Let again $W = \mathtt{GACAAAAACCCAT}$, $k = 3$, and $\mathcal{S} = \{\mathtt{ACA}, \mathtt{CAA}, \mathtt{AAA}, \mathtt{AAC}, \mathtt{CCA}\}$. Further, let the frequency threshold be $\tau = 2$. Note that the frequency of all non-sensitive length-3 patterns in $W$ is preserved in all three sanitized strings $X_{\mathrm{TR}} = \mathtt{GAC\#ACC\#CCC\#CAT}$, $X_{\mathrm{MIN}} = \mathtt{GACCC\#CAT}$, and $X_{\mathrm{ED}} = \mathtt{GAC\#AA\#ACCC\#CAT}$. Replacing, however, all #'s with G would create $\tau$-ghost GAC both in $X_{\mathrm{TR}}$ and in $X_{\mathrm{ED}}$.

**Contributions.** To our knowledge, there does not exist a general solution to the question we pose here that simultaneously guarantees effectiveness and efficiency. In this work, we provide compelling evidence as to why this is the case. Within the string sanitization context, we also provide algorithms for answering this question. Specifically:

**1)** We embark on a theoretical study to understand the relation between replacing #'s and creating $\tau$-ghosts. In particular, we define the following problems and examine their hardness:

- HMD (Hide and Mine decision): This is the core decision version of the problem asking whether or not we can replace all #'s in $X$, so that no sensitive pattern and no $\tau$-ghost occurs in $Z$. Deciding this may allow for sanitizing $X$ with no utility loss in frequent pattern mining. We show that HMD is *strongly NP-complete* via a reduction from a variant of the well-known Bin Packing problem [14] (see Section III). This is the most technically involved part of the paper, as the provided reduction is highly non-trivial.
- HM (Hide and Mine): This is the optimization version of HMD asking how we can replace all #'s, while ensuring that no sensitive patterns and a minimal number of $\tau$-ghosts occur in $Z$. This would minimize the utility loss in frequent pattern mining. HM is clearly NP-hard as a consequence of HMD being NP-complete, but we also show that it is *hard to approximate*.
- HMMT (Hide and Mine minimum threshold): Given a parameter $\tau$, this problem asks for the minimum frequency threshold $\tau_1 \geq \tau$ for which no sensitive pattern and no $\tau_1$-ghost occurs in $Z$. Solving HMMT would imply no utility loss in frequent pattern mining at a higher frequency threshold $\tau_1$ that is as close as possible to $\tau$. We show that HMMT is (*NP-hard* and) *hard to approximate*.

The hardness (see Section III) and inapproximabilty (see Section IV) results for our problems provide solid evidence for the lack of polynomial-time exact or approximation algorithms for these problems and motivate our next contribution.

**2)** We develop *exact algorithms* for HMD and HM (see Section V) that require polynomial time, under certain realistic assumptions on the problem parameters:

- Exact algorithms based on an Integer Linear Programming (ILP) formulation of HMD. The main idea is to identify all length-$k$ strings over $\Sigma$ in $X$ that may potentially become $\tau$-ghosts in $Z$, and then decide whether each of the #'s can be replaced by a letter in $\Sigma$ without creating any sensitive pattern or any $\tau$-ghost pattern in $Z$. We prove that HMD is *fixed-parameter tractable* [11] in most cases encountered in practice (e.g., when the number of distinct letters in the string and the length $k$ of sensitive patterns are upper bounded by a constant).
- Exact algorithms based on an ILP formulation of HM. This ILP formulation differs from the HMD formulation in that it takes into account the number of $\tau$-ghosts created by replacing #'s, so as to minimize their number. We prove that HM is fixed-parameter tractable in many cases encountered in practice (e.g., when the length $k$ of sensitive patterns and the number of distinct patterns that may become $\tau$-ghosts are upper bounded by a constant).

## II. Preliminaries and Problem Statement

An *alphabet* $\Sigma$ is a finite nonempty set whose elements are called *letters*. We also consider an alphabet $\Sigma_{\#} = \Sigma \cup \{\#\}$, where # is a special letter not in $\Sigma$. We fix a *string* $X = X[0] \cdots X[n-1]$ of length $|X| = n$ over $\Sigma_{\#}$. The set of length-$k$ strings over $\Sigma$ is denoted by $\Sigma^k$. For two indices $0 \leq i \leq j < n$, $X[i\,..\,j] = X[i] \cdots X[j]$ is the *substring* of $X$ that starts at position $i$ and ends at position $j$ of $W$. $\mathrm{Freq}_X(U)$ denotes the number of occurrences (starting positions) of string

$U$ as a substring of $X$. A *prefix* of $X$ is a substring of $X$ of the form $X[0\,..\,j]$, and a *suffix* of $X$ is a substring of $X$ of the form $X[i\,..\,n{-}1]$. A *dictionary* over $\Sigma$ is a set of strings over $\Sigma$. The dictionary used in our work is a set of length-$k$ strings that do not occur in $X$; we refer to these strings as *sensitive patterns*. Any element of $\Sigma^k$ that is not in this dictionary is referred to as a *non-sensitive pattern*. In combinatorics on words, such a dictionary is known as *antidictionary* and the sensitive patterns are known as *forbidden patterns* (e.g., see [10]).

**Problem 1** (HIDE & MINE (HM))**.** Given an integer $k > 0$, a string $X = X_0\#X_1\#\cdots\#X_\delta$ of length $n$ over an alphabet $\Sigma_\#$, with $|X_i| \geq k-1$, for all $i \in [0, \delta]$, a dictionary $\mathcal{S} \subseteq \Sigma^k$ such that no $S \in \mathcal{S}$ occurs in $X$, and an integer $\tau > 0$, compute a function $g : [\delta] \to \Sigma$ such that the following hold for string $Z = X_0 g(1) X_1 g(2) \cdots g(\delta) X_\delta$:

I The number of strings $U \in \Sigma^k$, with $\mathrm{Freq}_X(U) < \tau$ and $\mathrm{Freq}_Z(U) \geq \tau$ in $Z$, is minimized.
II No $S \in \mathcal{S}$ occurs in $Z$.

Note that function $g$ replaces each $\#$ by exactly one letter from $\Sigma$. Condition $|X_i| \geq k-1$ means that any two $\#$'s in $X$ are at least $k$ positions apart. Thus, any length-$k$ substring $X[i\,..\,i{+}k{-}1]$ of $X$ is affected by at most one $\#$ replacement. The sanitization method of [4, Lemma 1] produces an $X$ satisfying this condition, for *any* set $\mathcal{S} \subseteq \Sigma^k$, to guarantee that the frequency of every non-sensitive pattern is preserved in $X$. Thus, HM is directly applicable to the output of [4].

A string $U \in \Sigma^k$ with $\mathrm{Freq}_X(U) < \tau$ and $\mathrm{Freq}_Z(U) \geq \tau$ is referred to as $\tau$-*ghost*. To prove NP-completeness, we consider the decision variant HMD of HM, which asks to decide if there exists any function $g : [\delta] \to \Sigma$ such that the following hold:

I No $\tau$-ghost occurs in $Z$.
II No $S \in \mathcal{S}$ occurs in $Z$.

## III. HMD IS NP-COMPLETE

Problem HMD is clearly in NP. In this section, we show it to be strongly NP-complete via a reduction from a variant of Bin Packing [14].

### A. The UNIQUE-WEIGHTS BIN PACKING problem

The BIN PACKING (BP) problem is defined as follows. Given three positive integers, $M$ (number of bins), $B$ (capacity of every bin), and $N$ (number of items), and a vector $[w_1, \ldots, w_N]$ of positive integers (the weights of the items), BP asks whether we can partition the items into $M$ subsets (bins) without exceeding the capacity of any bin.

BP is *strongly* NP-complete [14], i.e., it is NP-complete even when weights and bin capacities are bounded by a polynomial function of $N$ and $M$. We can thus use gadgets whose size is proportional to the numerical values in the instance $\mathcal{I}_{\mathrm{BP}}$ of BP, as if we were representing those numbers in unary notation. To simplify the reduction, we assume there are no items of weight 1 (they can be added at the end where capacity is left), and that no two items have the same weight. We refer to this variant as UNIQUE-WEIGHTS BIN PACKING (UWBP). UWBP is also strongly NP-complete; we defer the proof of this claim to the full version of the paper.

**Lemma 1.** UWBP *is strongly NP-complete.*

### B. Overview of the Reduction from UWBP to HMD

For any UWBP instance, we construct in polynomial time an instance of HMD that has positive answer if and only if UWBP has positive answer. To this end, we will introduce several gadgets which will serve to model the different constraints of UWBP. Each gadget consists of a string of length $2k-1$ over a specific alphabet: $\#$, $x$, $y$, \$, and a letter $b_i$ for each $i \in [M]$. We will explain how all UWBP constraints are linked to the gadgets. The gadget $t_{ij}$ models whether item $j \in [N]$ is placed in bin $i \in [M]$:

$$t_{ij} = b_i \underbrace{x \ldots x}_{k-w_j-1} \underbrace{b_i \ldots b_i}_{w_j-1} \# \underbrace{b_i \ldots b_i}_{k-1}$$

The structure models the weight of items placed in bin $i$: when we replace the $\#$ with $b_i$, we introduce $w_j$ occurrences of $b_i^k$. The gadget $u_{ij}$, together with $t_{ij}$ and the set of forbidden patterns, ensures that each item is placed in some bin:

$$u_{ij} = b_i \underbrace{x \ldots x}_{k-w_j-1} \underbrace{b_i \ldots b_i}_{w_j-1} \# \underbrace{y \ldots y}_{w_j} \underbrace{x \ldots x}_{k-w_j-2} y$$

We link the filling of the $i$th bin with the number of occurrences of $b_i^k$. To limit the other non-sensitive patterns flexibly, we then choose a value $\tau$ high enough, and lower the available occurrences of each pattern by adding extra copies of them at the end. Namely, we have $k = \max_j w_j + 3$ and $\tau = \max\{M, B\} + 1$.

The final instance of HMD is the concatenation of the following patterns separated by the string \$\$:

1) $t_{ij}$, $\forall i, j$.
2) $u_{ij}$, $\forall i, j$.
3) $\tau - B - 1$ occurrences of $b_i^k$, $\forall i$ (allowed occurrences of $b_i^k$ model the capacity of bin $i$).
4) $\tau - 2$ occurrences of $b_i x^{k-w_j-1} b_i^{w_j-1} x$, $\forall i, j$ (only one more occurrence of this pattern is allowed, and one is created by replacing the $\#$ in $t_{ij}$ or $u_{ij}$ with $x$).
5) $\tau - M$ occurrences of $y^{w_j+1} x^{k-w_j-2} y$, $\forall j$ (allowed occurrences force us to replace at least one $u_{\cdot j}$ with $x$ for each $j$, thus forcing us to use $b_{ij}$ in the corresponding $t_{ij}$ gadget, i.e., placing each item in a bin).

The set $\mathcal{S}$ of sensitive patterns is carefully chosen to link these gadgets, and consists of the union of the following sets:

1) $\{b_{i'} b_i^{k-1} \mid i, i' \in [M], i' \neq i\}$, which forbids putting a $b_{i'}$ to replace the $\#$ in any $t_{ij}$, if $i' \neq i$.
2) $\{b_i y b_i^{k-2} \mid i \in [M]\}$, which forbids putting a $y$ to replace the $\#$ in a $t_{ij}$.
3) $\{b_i \$ b_i^{k-2} \mid i \in [M]\}$, which forbids putting a \$ to replace the $\#$ in a $t_{ij}$.
4) $\{b_i y^{w_j} x^{k-w_j-2} y \mid i \in [M], j \in [N]\}$, which forbids putting any $b_i$ to replace the $\#$ in a $u_{ij}$.

5) $\{b_i\$y^{w_j}x^{k-w_j-2} \mid i \in [M], j \in [N]\}$, which forbids putting a \$ to replace the # in a $u_{ij}$.

It can be shown that this instance of HMD has positive answer if and only if the original UWBP does, thus proving our claim. We defer the details to the full version of the paper.

**Theorem 1.** HMD *is strongly NP-complete.*

## IV. HM is Hard to Approximate

Given the hardness of HMD, we now shift our focus on checking whether an approximately optimal solution of HM can be obtained instead. Given any instance $\mathcal{I}_M$ of a minimization problem $M$, an algorithm is called an $\alpha$-*approximation*, for some $\alpha \geq 1$, if it runs in polynomial time in the size of $\mathcal{I}_M$ and always outputs a solution value $\Gamma \leq \alpha \cdot \text{OPT}$, where OPT denotes the optimal value for $\mathcal{I}_M$. We start with the following:

**Theorem 2.** *There is no $\alpha$-approximation algorithm for* HM*, for any $\alpha \geq 1$, unless P=NP.*

*Proof.* Suppose by contradiction that an $\alpha$-approximation algorithm $A$ existed for minimizing the number of $\tau$-ghosts in HM. We could then use $A$ to solve HMD: the answer to HMD would be positive (i.e., there would exist a function $g$ that creates 0 $\tau$-ghosts) if and only if the answer of $A$ was $\Gamma = 0 \leq \alpha \cdot \text{OPT} = 0$, which contradicts Theorem 1. □

The reader may now wonder whether the problem becomes easier should one relax the requirement for a *fixed threshold* $\tau$. Thus, the following problem arises naturally.

**Problem 2** (HMMT)**.** Given an integer $k > 0$, a string $X = X_0\#X_1\#\cdots\#X_\delta$ of length $n$ over alphabet $\Sigma_\#$, with $|X_i| \geq k-1$ for all $i \in [0, \delta]$, a dictionary $\mathcal{S} \subseteq \Sigma^k$ such that no $S \in \mathcal{S}$ occurs in $X$, and an integer $\tau_0 > 0$, compute the smallest integer $\tau_1 \geq \tau_0$ so that there exists a function $g : [\delta] \to \Sigma$, such that the following hold for string $Z = X_0 g(1) X_1 g(2) \cdots g(\delta) X_\delta$:

I No $U \in \Sigma^k$, with $\text{Freq}_X(U) < \tau_1$ and $\text{Freq}_Z(U) \geq \tau_1$ occurs in $Z$.

II No $S \in \mathcal{S}$ occurs in $Z$.

The practical rationale for considering HMMT is that it could be useful if, for instance, $\tau_1$ is only slightly larger than $\tau$ in a given HM instance. Unfortunately, we show that HMMT is NP-hard, and it is even hard to approximate.

**Theorem 3.** HMMT *is NP-hard.*

*Proof.* We reduce HMD to HMMT as follows. Let $\mathcal{I}_{\text{HMD}}$ be the instance of HMD we would like to solve for some threshold $\tau$. We construct an instance of HMMT consisting of the $X$, $k$, and $\mathcal{S}$ from $\mathcal{I}_{\text{HMD}}$, and we also set $\tau_0 = \tau$. We denote this instance by $\mathcal{I}_{\text{HMMT}}$. The reduction takes linear time in the size of HMD. We seek to find the minimum threshold $\tau_1 \geq \tau_0$ such that no length-$k$ substring of $Z$ is a $\tau_1$-ghost. Then $\mathcal{I}_{\text{HMD}}$ has a positive answer if and only if the answer $\tau_1$ of $\mathcal{I}_{\text{HMMT}}$ is equal to $\tau_0 = \tau$. The statement thus follows. □

Observe that a pattern $U$ is a $\tau$-ghost if and only if $\tau \in (\text{Freq}_X(U), \text{Freq}_Z(U)]$. Therefore, the minimal number of $\tau$-ghosts is not monotonous in $\tau$. On the contrary, the minimal number of $\tau$-ghosts is zero when $\tau = 0$ and all patterns are already frequent (i.e., they appear at least $\tau$ times), or when $\tau > n$ and the threshold is so high that no pattern can ever become a $\tau$-ghost. In between, the minimal number of $\tau$-ghosts increases whenever $\tau$ equals the frequency of some patterns in $X$, and then slowly decreases again. We will use this behavior, and the fact that HMD is NP-hard, to construct a string for which we cannot determine in polynomial time whether $\tau_1 = \tau_0$ or $\tau_1 > \alpha\tau_0$ (and for which we can prove that $\tau_1 \notin [\tau_0 + 1, \alpha\tau_0]$), implying inapproximability.

**Theorem 4.** *There is no $\alpha$-approximation algorithm for* HMMT*, for any $\alpha \geq 1$, unless P=NP.*

*Proof.* Let $X$ be an arbitrary string and $\mathcal{S}$ be the set of sensitive patterns as defined in HMD. Further, let $T$ be the length-$(k-2)$ suffix of $X$ and $Z$ be a string obtained by replacing the #'s of $X$. From this instance of HMD, we will construct an instance of HMMT consisting of a string $Y$ and a set $\mathcal{S}'$ of sensitive patterns, so that if an $\alpha$-approximation algorithm existed for HMMT, we could decide HMD in polynomial time. We define $Y$ over $\Sigma \cup \{\#, \&\}$ to be

$$Y = X(\&\&T)^{\tau_0}\&(\#T\&)^{\lceil(\alpha-1)\tau_0\rceil}.$$

Let $\mathcal{R}$ be the set of all strings $\&sT$, with $s \in \Sigma$. We define the dictionary of sensitive patterns be $\mathcal{S}' = \mathcal{S} \cup \mathcal{R}$. Note that we need to replace all #'s in $(\#T\&)^{\lceil(\alpha-1)\tau_0\rceil}$ by &'s in order not to introduce any sensitive patterns. However, doing so increases the number of $\&T\&$ patterns (and all other newly created patterns) from $\tau_0$ to $\lceil\alpha\tau_0\rceil$. Therefore, if $\tau = \tau_0$, then the number of $\tau$-ghosts in $Z$ equals that in $Z(\&\&T)^{\tau_0}\&(\&T\&)^{\lceil(\alpha-1)\tau_0\rceil}$, because the additional new patterns were already occurring at least $\tau$ times in $Y$. However if $\tau_0 < \tau \leq \lceil\alpha\tau_0\rceil$, then there will always be at least one $\tau$-ghost, namely $\&T\&$. Recall that deciding HMD is NP-complete. Therefore it is NP-complete to decide whether or not $\tau_1 = \tau_0$ or $\tau_1 > \lceil\alpha\tau_0\rceil$. We conclude that there exists no $\alpha$-approximation algorithm for HMMT, unless P=NP. □

## V. Exact Algorithms for HM

We resort to ILP to design exact algorithms for HMD and HM. In particular, we show that both problems are fixed-parameter tractable for several combinations of parameters.

We say that the length-$(k-1)$ substring $U$ preceding an occurrence of # in $X$, and the length-$(k-1)$ substring $V$ following it, form its *context* $UV$. Recall that there are $\delta$ occurrences of # in $X$, and that any two occurrences are at least $k$ letters apart, so $UV$ is in $\Sigma^{2k-2}$. We assign to every context $UV$ a unique identifier (id). We write $\#_i$ for # in $X$ if its context $UV$ has id $i$. A string $N \in \Sigma^k$ is *critical* if it may become a $\tau$-ghost, i.e., if an additional occurrence of $N$ can be created by replacing some # by a letter in $\Sigma$ and $\text{Freq}_X(N) \in [\tau - k\delta, \tau - 1]$. This is because the frequency of $N$ cannot increase by more than $k\delta$, and the frequency of $N$

in $X$ must be less than $\tau$ for $N$ to become $\tau$-ghost. We assign to each critical string $N$ a unique id $\ell$, and denote it by $N_\ell$. We introduce the following parameters:

- $\boldsymbol{\gamma}$ number of distinct contexts present in $X$;
- $\boldsymbol{\delta_i}$ number of occurrences of letter $\#_i$ in $X$, for $i \in [\gamma]$;
- $\boldsymbol{\lambda}$ number of distinct critical length-$k$ strings;
- $\boldsymbol{\alpha^i_{\ell,j}}$ additional number of occurrences of $N_\ell$ introduced by replacing a $\#_i$ with a letter $j \in \Sigma$, for $\ell \in [\lambda]$;
- $\boldsymbol{e_\ell}$ difference $(\tau - 1) - \mathrm{Freq}_X(N_\ell)$, for $\ell \in [\lambda]$.

Intuitively, $\boldsymbol{e_\ell}$ is the *budget* we have for $N_\ell$: the number of its additional occurrences we can afford. Since replacing an occurrence of $\#_i$ by $j \in \Sigma$ adds $k$ new strings in $\Sigma^k$, $\boldsymbol{\alpha^i_{\ell,j}}$ counts how many of them are equal to $N_\ell$. Let $\boldsymbol{x_{i,j}}$ be the number of times we replace $\#_i$ by $j \in \Sigma$, and let $\mathcal{F} \subseteq [\gamma] \times \Sigma$ be the set of *forbidden* replacements: $(i,j) \in \mathcal{F}$ if and only if replacing $\#_i$ by $j$ introduces a sensitive pattern. To determine whether there exists a way of replacing all $\#$'s with letters without introducing any sensitive patterns nor $\tau$-ghosts, we need to find a solution $x \in \mathbb{Z}^{\gamma \times |\Sigma|}$ to the following problem:

$$\begin{cases} x_{i,j} \geq 0 & \forall (i,j) \in [\gamma] \times \Sigma \\ x_{i,j} = 0 & \forall (i,j) \in \mathcal{F} \\ \sum_{i \in [\gamma], j \in \Sigma} \alpha^i_{\ell,j} x_{i,j} \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j \in \Sigma} x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{1}$$

The first and fourth constraints ensure that each $\#$ is replaced by exactly one letter, the second constraint that we do not reinstate any sensitive patterns, and the third constraint that we do not introduce any $\tau$-ghosts. This is clearly an ILP with $m = \gamma|\Sigma|$ variables and at most $2m+\lambda+\gamma$ constraints. The well-known algorithm by Megiddo [23] solves the ILP problem in linear time in the number constraints (resp. variables) when the number of variables (resp. constraints) is upper bounded by a constant. Hence, although HMD is NP-complete in general, if appropriate subsets of parameters are bounded by a constant, we can count on polynomial-time solutions.

To show that HMD takes polynomial time in certain cases, let us start with a general preprocessing step. We construct a static dictionary with $\mathcal{O}(1)$ access time of the letters in $X$ and the letters in strings of $\mathcal{S}$. The value (id) of each key (letter) is chosen from $\{1, \ldots, k|\mathcal{S}| + n\}$. This construction can be done in $\mathcal{O}(n + k|\mathcal{S}|)$ time using perfect hashing [13]. We can thus lexicographically sort all length-$k$ substrings of $X$ and all length-$k$ strings in $\mathcal{S}$ (viewed as strings over letter id's) using radix sort in $\mathcal{O}(nk+|\mathcal{S}|k)$ time, and construct two dictionaries, one for $X$ and one for $\mathcal{S}$, as follows. For $X$, we construct a trie of all its non-sensitive length-$k$ substrings. The value of each key (non-sensitive pattern) is its multiplicity in $X$. We also construct a trie of all strings in $\mathcal{S}$ in a similar fashion (no multiplicities are relevant here, so no values are stored). We store in both tries, for every node, the first letter on each of its outgoing edges in a static dictionary with $\mathcal{O}(1)$ access time [13]. Thus both trie dictionaries support $\mathcal{O}(k)$ access time: if a length-$k$ string $Q$ is given as a query, we first convert it to a string $I(Q)$ of id's in $\mathcal{O}(k)$ time using the letter dictionary, and then search for $I(Q)$ from the root of the tries in $\mathcal{O}(k)$ time. The total construction time is $\mathcal{O}(nk+|\mathcal{S}|k)$.

When $\delta = \mathcal{O}(1)$, the brute-force algorithm checking all possible ways to replace the $\#$'s with letters of $\Sigma$ runs in polynomial time. There are $|\Sigma|^\delta$ ways to replace the $\#$'s. Each of these ways generates $\delta k$ new length-$k$ strings for which we have to check if they are sensitive or create a $\tau$-ghost. Checking if they are sensitive can be done using the trie of $\mathcal{S}$ in $\mathcal{O}(k)$ time per each length-$k$ string. Counting the additional number of occurrences of each length-$k$ substring of $X$ can be done using the trie of $X$ in $\mathcal{O}(k)$ time. Counting the number of occurrences of each length-$k$ string that does not occur in $X$ can be done by constructing a trie of all such strings (we have at most $\delta k$ of them per way), similar to the preprocessing step. This gives $\mathcal{O}(nk + |\mathcal{S}|k + |\Sigma|^\delta \delta k^2)$ time in total.

A problem with parameters $p$ and $q$ is *fixed-parameter tractable* (FPT) in $p$ if there exists a function $f$ and a polynomial $P$ such that the problem has time complexity $\mathcal{O}(f(p) \cdot P(q))$ [11]. The following theorem shows three scenarios where an FPT algorithm exists for HMD.

**Theorem 5.** HMD *is fixed-parameter tractable if*

- $(a)$ $|\Sigma| = \mathcal{O}(1)$ *and* $\gamma = \mathcal{O}(1)$*; or*
- $(b)$ $|\Sigma| = \mathcal{O}(1)$ *and* $k = \mathcal{O}(1)$*; or*
- $(c)$ $k = \mathcal{O}(1)$ *and* $\lambda = \mathcal{O}(1)$.

*Proof.* We first perform the above-mentioned preprocessing. $(a)$ We will solve this case by constructing and solving the ILP in Eq. 1. We can count the number of occurrences of each length-$k$ substring of $X$ using the trie of $X$ (and thus determine $e_\ell$ for these strings) in $\mathcal{O}(nk)$ time. The id $i$ of each context $\#_i$ and its number $\delta_i$ of occurrences can be determined within the same complexity using a similar preprocessing: this is possible because the length of every context is $2k - 2 = \mathcal{O}(k)$. Finally, the $\alpha^i_{\ell,j}$'s and $\mathcal{F}$ can be computed in $\mathcal{O}(\gamma|\Sigma|k^2)$ total time as follows. For a context $\#_i$ and a letter $j \in \Sigma$, we create $k$ new length-$k$ strings when replacing $\#_i$ with $j$, each of which is either sensitive (in which event we add $(i,j)$ to $\mathcal{F}$) or non-sensitive (we increase $\alpha^i_{\ell,j}$ by 1). Checking if they are sensitive can be done using the trie of $\mathcal{S}$ in $\mathcal{O}(k)$ time per length-$k$ string. Counting the additional number of occurrences of a critical length-$k$ substring of $X$ can be done using the trie of $X$ in $\mathcal{O}(k)$ time. Counting the number of occurrences of a critical length-$k$ string that does not occur in $X$ (note that $e_\ell = \tau - 1$ for these strings) can be done by constructing a trie of all such strings, similar to the preprocessing step. The ILP is thus constructed in $\mathcal{O}(nk + |\mathcal{S}|k + \gamma|\Sigma|k^2)$ total time. Since the number of variables in the ILP is $m = \gamma|\Sigma| = \mathcal{O}(1)$ and solving ILP's is fixed-parameter linear in the number of variables [23], HMD is FPT if $\gamma$ and $|\Sigma|$ are fixed.

$(b)$ Since every context has length $2k - 2$ and also $|\Sigma| = \mathcal{O}(1)$ and $k = \mathcal{O}(1)$, we have that $\gamma \leq |\Sigma|^{2k-2} = \mathcal{O}(1)$. Thus, if $k$ and $|\Sigma|$ are fixed, we are in case $(a)$, and HMD is FPT.

$(c)$ If $k = \mathcal{O}(1)$ and $\lambda = \mathcal{O}(1)$, the numbers of constraints and variables in the ILP are not necessarily upper bounded by a constant. Therefore, we cannot directly solve the ILP in polynomial time. However, since the $\lambda$ critical length-$k$ strings contain overall at most $\lambda k$ different letters, we actually only need to distinguish among a bounded number of letters.

Since we do not need to consider explicitly the remaining letters, we rather represent them by a single special letter. Let $\sigma \subseteq \Sigma$ denote the set of letters contained in critical length-$k$ strings. Note that critical length-$k$ strings can be determined as described in part $(a)$. Thus $\sigma$ can be specified and indexed using perfect hashing [13] within the same time complexity. We introduce a new letter \$ representing all the letters in $\Sigma \setminus \sigma$, and we denote by $\mathcal{F}|_{\$}$ the set of forbidden replacements where all pairs $(i,j) \in \mathcal{F}$ with $j \in \Sigma \setminus \sigma$ are collapsed in a single pair $(i, \$)$. We thus need to find a solution $x \in \mathbb{Z}^{\gamma \times (|\sigma|+1)}$ for:

$$\begin{cases} x_{i,j} \geq 0 & \forall i \in [\gamma], j \in \sigma \cup \{\$\} \\ x_{i,j} = 0 & \forall (i,j) \in \mathcal{F}|_{\$} \\ \sum_{i \in [\gamma], j \in \sigma} \ \alpha^i_{\ell,j} x_{i,j} \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j \in \sigma \cup \{\$\}} \ x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{2}$$

This new ILP can be constructed in $\mathcal{O}(nk + |\mathcal{S}|k + \gamma|\Sigma|k^2)$ time, like Eq. 1. Since the ILP has only $\gamma(|\sigma|+1) = \mathcal{O}(1)$ variables, HMD is FPT for fixed $k$ and $\lambda$ [23]. We can obtain a solution to the original problem by replacing \$ by any letter in $\Sigma \setminus \sigma$ that does not create a sensitive pattern. □

We can decide in polynomial time if HM has a solution: we check all $|\Sigma|$ letter replacements at each of the $\delta$ positions where a # occurs. If, at each position, there exists at least one letter replacement that does not create a sensitive pattern, then HM has a solution. Thus, without loss of generality we assume that HM always has a solution. To minimize $\tau$-ghosts in $Z$, we define a binary variable $z_\ell$, $\ell \in [\lambda]$, which is equal to 1 (resp. 0) when $N_\ell$ has (resp. has not) become $\tau$-ghost. The ILP formulation for HM is to find $x \in \mathbb{Z}^{\gamma \times |\Sigma|}$ so as to: Minimize $\sum_{\ell=1}^{\lambda} z_\ell$ subject to

$$\begin{cases} x_{i,j} \geq 0 & \forall (i,j) \in [\gamma] \times \Sigma \\ x_{i,j} = 0 & \forall (i,j) \in \mathcal{F} \\ z_\ell \geq 0 & \forall \ell \in [\lambda] \\ \sum_{i \in [\gamma], j \in \Sigma} \ \alpha^i_{\ell,j} x_{i,j} - k\delta z_\ell \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j \in \Sigma} \ x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{3}$$

Note that, in the ILP of Eq. 3, $\sum_{i \in [\gamma], j \in \Sigma} \alpha^i_{\ell,j} x_{i,j} - k\delta z_\ell \leq e_\ell$ if and only if $N_\ell$ is not a $\tau$-ghost or $z_\ell = 1$.

**Theorem 6.** HM *is fixed-parameter tractable if*

$(a)$ $|\Sigma| = \mathcal{O}(1)$, $\gamma = \mathcal{O}(1)$, *and* $\lambda = \mathcal{O}(1)$*; or*

$(b)$ $k = \mathcal{O}(1)$ *and* $\lambda = \mathcal{O}(1)$.

*Proof.* $(a)$ We can obtain the ILP of Eq. 3 in $\mathcal{O}(\lambda)$ time from the ILP of Eq. 1, which can be constructed in $\mathcal{O}(nk + |\mathcal{S}|k + \gamma|\Sigma|k^2)$ time; see the proof of Theorem 5(a). The ILP of Eq. 3 has at most $2m + 2\lambda + \gamma$ constraints and $m + \lambda = |\Sigma|\gamma + \lambda$ variables. Therefore HM is FPT if $|\Sigma|$, $\gamma$ and $\lambda$ are fixed [23].

$(b)$ Similar to the ILP of Eq. 2 (see Theorem 5(c)), we can reduce the alphabet $\Sigma$ to the letters of the critical length-$k$ strings and a special letter \$. This new minimization ILP has $\gamma(|\sigma|+1)+\lambda \leq (k\lambda+1)^{2k-1}+\lambda = \mathcal{O}(1)$ variables. Therefore HM is FPT if $k$ and $\lambda$ are fixed [23]. □

**Acknowledgments.** MIUR Grant 20174LF3T8 AHeAD; University of Pisa "PRA – Progetti di Ricerca di Ateneo" (Institutional Research Grants) Grant PRA_20202021_26 "Metodi Informatici Integrati per la Biomedica"; and NWO Gravitation-grant NETWORKS-024.002.003.

## REFERENCES

[1] O. Abul, F. Bonchi, and F. Giannotti. Hiding sequential and spatiotemporal patterns. *TKDE*, 22(12):1709–1723, 2010.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995.

[3] H. Arimura and T. Uno. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. *J. Comb. Optim.*, 13(3):243–262, 2007.

[4] G. Bernardini, H. Chen, A. Conte, R. Grossi, G. Loukides, N. Pisanti, S. P. Pissis, and G. Rosone. String sanitization: A combinatorial approach. In *ECML/PKDD*, pages 627–644, 2019.

[5] G. Bernardini, H. Chen, G. Loukides, N. Pisanti, S. P. Pissis, L. Stougie, and M. Sweering. String sanitization under edit distance. In *CPM*, pages 7:1–7:14, 2020.

[6] E. Bier, R. Chow, P. Golle, T. H. King, and J. Staddon. The rules of redaction: Identify, protect, review (and repeat). *IEEE Secur. Priv.*, 7(6):46–53, 2009.

[7] F. Biessmann, D. Salinas, S. Schelter, P. Schmidt, and D. Lange. "deep" learning for missing value imputationin tables with non-numerical data. In *CIKM*, pages 2017–2025, 2018.

[8] M. Chen, X. Yu, and Y. Liu. Mining moving patterns for predicting next location. *Inf. Syst.*, 54(C):156–168, 2015.

[9] N. Cristianini and M. W. Hahn. *Introduction to computational genomics - a case studies approach*. Cambridge University Press, 2007.

[10] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. Text compression using antidictionaries. In *ICALP*, pages 261–270, 1999.

[11] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

[12] C. Fiot, A. Laurent, and M. Teisseire. Approximate sequential patterns for incomplete sequence database mining. In *FUZZ*, pages 1–6, 2007.

[13] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with O(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[14] M. R. Garey and D. S. Johnson. "Strong" NP-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, 1978.

[15] A. Gkoulalas-Divanis and G. Loukides. Revisiting sequential pattern hiding to enhance utility. In *KDD*, pages 1316–1324, 2011.

[16] J. W. Grzymala-Busse and M. Hu. A comparison of several approaches to missing attribute values in data mining. In *Rough Sets and Current Trends in Computing*, pages 378–385, 2001.

[17] V. Guralnik and G. Karypis. A scalable algorithm for clustering sequential data. In *ICDM*, pages 179–186, 2001.

[18] IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970.

[19] U. Keich and P. A. Pevzner. Finding motifs in the twilight zone. *Bioinformatics*, 18(10):1374–1381, 2002.

[20] D. C. Koboldt, K. M. Steinberg, David E. Larson, Richard K. Wilson, and Elaine R. Mardis. The next-generation sequencing revolution and its impact on genomics. *Cell*, 155(1):27–38, 2013.

[21] R. Li, C. Yu, Y. Li, T. Wah Lam, S. Yiu, K. Kristiansen, and J. Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[22] H. M. Martinez. An efficient method for finding repeats in molecular sequences. *Nucleic Acids Research*, 11(13):4629–4634, 1983.

[23] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984.

[24] S. Rangavittal, R. S. Harris, M. Cechova, M. Tomaszkiewicz, R. Chikhi, K. D. Makova, and P. Medvedev. RecoverY: k-mer-based read classification for Y-chromosome-specific sequencing and assembly. *Bioinformatics*, 34(7):1125–1131, 2017.

[25] W. Shen, J. Wang, and J. Han. Sequential pattern mining. In C. C. Aggarwal and J. Han, editors, *Frequent Pattern Mining*, pages 261–282. 2014.

[26] M. Spiliopoulou. Managing interesting rules in sequence mining. In *PKDD*, pages 554–560, 1999.

[27] Y. Wu, C. Chiang, and A. L. P. Chen. Hiding sensitive association rules with limited side effects. *TKDE*, 19(1):29–42, 2007.

[28] J. J. Ying, W. Lee, T. Weng, and V. S. Tseng. Semantic trajectory mining for location prediction. In *SIGSPATIAL*, pages 34–43, 2011.