Hide and Mine in Strings: Hardness, Algorithms, and Experiments

Giulia Bernardini, Alessio Conte, Garance Gourdel, Roberto Grossi, Grigorios Loukides, *Member, IEEE*, Nadia Pisanti, Solon P. Pissis, Giulia Punzi, Leen Stougie, Michelle Sweering

Abstract—Data sanitization and frequent pattern mining are two well-studied topics in data mining. Data sanitization is the process of disguising (hiding) confidential information in a given dataset. Typically, this process incurs some utility loss that should be minimized. Frequent pattern mining is the process of obtaining all patterns occurring frequently enough in a given dataset. Our work initiates a study on the fundamental relation between data sanitization and frequent pattern mining in the context of sequential (string) data. Current methods for string sanitization hide confidential patterns. This, however, may lead to spurious patterns that harm the utility of frequent pattern mining. The main computational problem is to minimize this harm. Our contribution here is as follows. 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 realistic assumptions on the input parameters. We also complement the integer linear programming algorithms with a greedy heuristic. Third, we present an extensive experimental study, using both synthetic and real-world datasets, that demonstrates the effectiveness and efficiency of our methods. Beyond sanitization, the process of missing value replacement may also lead to spurious patterns. Interestingly, our results apply in this context as well. We show that, unlike popular approaches, our methods can fill missing values in genomic sequences, while preserving the accuracy of frequent pattern mining.

Index Terms—Data privacy, Data sanitization, Knowledge hiding, Frequent pattern mining, String algorithms

1 Introduction

A STRING is a sequence of letters over some alphabet Σ. 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 [1], 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 [3]. Mining patterns from such strings is thus useful in a gamut of applications: mining patterns from location history data helps route planning [4]; mining patterns of co-purchased products from market-basket data improves business decision making [2]; mining patterns from genome sequences can improve clinical diagnostics [3]. To support these applications while preserving privacy, strings representing individuals' data are often being disseminated after sanitization [5], [6] or anonymization [7].

- *G. Bernardini is with the University of Trieste, Italy and CWI, The Netherlands. Email: giulia.bernardini@units.it*
- *A. Conte and G. Punzi are with Università di Pisa, Italy. Email: conte@di.unipi.it, giulia.punzi@phd.unipi.it*
- *G. Gourdel is with Inria Rennes, École normale supérieure, ENS Paris-Saclay, France and Università di Pisa, Italy. Email: garance.gourdel@irisa.fr*
- *R. Grossi and N. Pisanti are with Università di Pisa, Italy and the ERABLE Team, France. Email:{grossi,pisanti}@di.unipi.it*
- *G. Loukides is with King's College London, United Kingdom. Email: grigorios.loukides@kcl.ac.uk*
- *S. P. Pissis and L. Stougie are with CWI and the Vrije Universiteit, The Netherlands, and with the ERABLE Team, France. Email: {solon.pissis, leen.stougie}@cwi.nl*
- *M. Sweering is with CWI, The Netherlands. Email: michelle.sweering@cwi.nl*

Manuscript received...

In this paper, we study the fundamental relation between *data sanitization* [5], [6], [8] (also known as *knowledge hiding*) and *frequent pattern mining* [9], [10], [11], [12]. The objective of frequent pattern mining in strings is to obtain all patterns occurring frequently enough (according to a given frequency threshold τ) 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 [13], [14]). In string sanitization, an adversary seeks to determine whether one or more *sensitive patterns* modeling confidential knowledge occur in (the sanitized version of) a string. For example, an adversary may want to determine whether a series of purchased products (resp. search queries) indicating pregnancy occur in a user's purchasing history (resp. search history) [15]. The adversary knows only the sanitized version of a string, the alphabet Σ over which the string is derived, and a set of sensitive patterns. The adversary succeeds if, based on their knowledge, they can determine whether one or more sensitive patterns occur in the string. In our example, the adversary's success would allow them to infer that a user is pregnant and potentially use this information in unsolicited advertisement [15]. The privacy objective of string sanitization is to negate the adversary's success criterion [8], [16], [17]. Of note, the adversary model and privacy objective of string sanitization is similar to that of itemset [5], [18] or sequence [19], [20] sanitization.

Let W be the input string over Σ, k be a positive integer, and $\mathbf{S}$ be the set of sensitive length-k substrings. Recently proposed methods [8], [16], [17] construct a string X satisfying the following properties: (I) X contains no element of $\mathbf{S}$ as a substring; (II) the total order and thus the frequency of all non-sensitive length-k substrings of W is preserved in

X, or a partial order of these substrings and their frequency is preserved in X; and (III) the length of X is minimized [8], or the edit distance between W and X is minimized [16], [17]. These methods work by copying carefully selected substrings of W into X and separating them by a special letter $\# \notin \Sigma$. Clearly, the privacy objective (i.e., property I) may be achieved by removing some letters from the sensitive patterns occurring in W, or by concatenating the non-sensitive substrings in W and separating them by $\#$. Yet, the first strategy is ineffective at preserving the utility of the string as it incurs large changes to the set of length-k frequent substrings [8], [16], while the second one leads to an unnecessarily long string that has a negative impact on the efficiency of any subsequent analysis tasks [8], [16]. On the other hand, the methods in [8], [16], [17] satisfy property I; ensure no accuracy loss in sequentiality-based or frequency-based tasks (e.g., that the same length-k frequent substrings can be mined from W and from X) due to property II; and help the subsequent analysis on X in terms of efficiency [8], [16] or utility [17] due to property III. Furthermore, they are efficient (i.e., they match or are close to the time-complexity lower bounds).

Example 1. Let $W = \texttt{GACAAAAACCCAT}$, $k = 3$, and the set of sensitive patterns $\mathbf{S} = \{\texttt{ACA}, \texttt{CAA}, \texttt{AAA}, \texttt{AAC}, \texttt{CCA}\}$. 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 substrings of W: X_{TR} is the trivial solution of interleaving the non-sensitive length-3 substrings of W with $\#$; X_{MIN} is the *shortest* possible such string [8]; and X_{ED} is a string *closest* to W in terms of edit distance [17].

Unfortunately, as noted in [8], the occurrences of $\#$ reveal the *locations* of sensitive patterns. Thus, an adversary who knows how $\#$'s are added to X, in addition to knowing X, Σ, and $\mathbf{S}$, can infer the sensitive patterns in X. To prevent this, the occurrences of $\#$'s must be ultimately replaced by letters of the original alphabet Σ. This replacement gives rise to another string over Σ, which we denote by Z. The replacement must ensure that sensitive patterns, as well as any implausible patterns (i.e., known or likely artefacts of sanitization that could be exploited to locate the positions of replaced $\#$'s), do not occur in Z (see [16] for details). However, Z may contain spurious patterns that could not be mined from X at a minimum frequency threshold τ but would be mined from Z at the same frequency threshold. These patterns are referred to as τ-*ghosts*.

Motivated by the importance of string sanitization and the useful properties of the methods of [8], [16], [17], we investigate the crucial interplay between $\#$ replacements and τ-ghosts. We pose 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 τ, how should we replace the $\#$'s in X with letters from Σ, so that the number of length-k τ-ghosts in the resulting string Z is minimized?* Answering this question helps preserving the accuracy of frequent pattern mining and tasks based on it (e.g., pattern-based clustering [21] and classification [22], as well as sequential rule mining [23]) that we may not know a priori. For example, in the context of data sanitization, answering this question would enable the mining of frequent patterns that model useful information about individuals (trips in location sequences, co-purchased products in market-basket sequences, or motifs in genomic sequences), as well as the protection of individuals' confidential information (certain location sequences, co-purchased products or parts of genome) [6], [8]. In addition, it would allow an organization to share sales data (e.g., a string in which a letter denotes the sale of a product) with a third party for collaboration purposes, without enabling the mining of information that could provide competitive advantage to the third party (e.g., a sequence of products that are sold unexpectedly frequently) [20].

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 (i.e., a *missing value*) from these datasets. For instance, in genome sequencing data, $\#$ corresponds to an unknown DNA base [24]; in databases, $\#$ represents a value that has not been recorded [25], [26]; and in masked data outputted by other privacy-preserving methods [27], $\#$ 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 compactly 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 $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}, \#\}$. 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., [28], [29]), 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 many spurious patterns, including patterns that are unlikely to occur in a genomic sequence [30], [31], negatively affecting the accuracy of frequent pattern mining. This is in contrast to our approach, which aims to replace unknown bases (occurrences of $\#$) in a way that avoids these patterns to preserve data utility (see Section 9 for further details).

Replacing the occurrences of $\#$ in a database is also often needed to be able to perform frequent pattern mining with off-the-shelf algorithms [26]. To this end, the occurrences of $\#$ are commonly replaced by some statistical estimate, such as the most frequent value [26], [32]. However, such a replacement does not generally maintain the accuracy of frequent pattern mining, since it may introduce many spurious patterns [26]. The goal of our approach is to preserve as much as possible the accuracy of frequent pattern mining, by minimizing the creation of spurious frequent patterns.

Example 2. Let again $W = \texttt{GACAAAAACCCAT}$, $k = 3$, and $\mathbf{S} = \{\texttt{ACA}, \texttt{CAA}, \texttt{AAA}, \texttt{AAC}, \texttt{CCA}\}$. Further, let the frequency threshold be $\tau = 2$. Note that the frequency of all non-sensitive patterns (length-3 substrings) in W is preserved

in all three sanitized strings $X_{\mathrm{TR}} = \texttt{GAC\#ACC\#CCC\#CAT}$, $X_{\mathrm{MIN}} = \texttt{GACCC\#CAT}$, and $X_{\mathrm{ED}} = \texttt{GAC\#AA\#ACCC\#CAT}$. Replacing, however, all #'s with G would create τ-ghost GAC both in X_{TR} and in X_{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. 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 τ-ghosts. In particular, we define the following problems, which all require that any two #'s in X are at least k positions apart, 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 τ-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 [33] (see Section 4). 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 τ-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 τ, this problem asks for the minimum frequency threshold $\tau_1 \geq \tau$ for which no sensitive pattern and no τ_1-ghost occurs in Z. Solving HMMT would imply no utility loss in frequent pattern mining at a higher frequency threshold τ_1 that is as close as possible to τ. We show that HMMT is (*NP-hard* and) *hard to approximate*.

The hardness (see Section 4) and inapproximabilty (see Section 5) results for our problems provide solid evidence for the lack of exact or approximation polynomial-time algorithms for these problems (also for the generalized problem, in which there are no restrictions on the distance between #'s), and motivate our next contributions. These results are general and independent of the application for which #'s are replaced. In particular, they rigorously answer how difficult is to apply frequent pattern mining and missing value replacement from the lower bound point of view.

2) We develop *exact algorithms* for HMD and HM that require polynomial time, under certain realistic assumptions on the problem parameters. We also develop an efficient and effective heuristic for HM. In particular, we develop the following:

- Exact algorithms based on an Integer Linear Programming (ILP) formulation of HMD. The main idea is to identify all length-k strings over Σ in X that may potentially become τ-ghosts in Z, and then decide whether each of the #'s can be replaced by a letter in Σ without creating any τ-ghost pattern or any sensitive pattern in Z. We prove that HMD is *fixed-parameter tractable*[1] 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 τ-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 τ-ghosts are upper bounded by a constant).
- A greedy heuristic that replaces the #'s from left to right, while avoiding the creation of non-sensitive patterns that may become τ-ghosts. The heuristic has three variants which aim to minimize different measures based on: the number of newly created patterns with frequency $f < \tau$, the sum of $(\tau - f)^{-1}$, or the max of $(\tau - f)^{-1}$, where frequency f is taken over the newly created patterns.

The ILP-based algorithms are presented in Section 6, and the greedy heuristic in Section 7.

3) We conduct an extensive experimental study (see Section 8). We show that our methods: (I) allow for frequent length-k pattern mining with no or insignificant utility loss (i.e., they create zero or few τ-ghosts); (II) incur very low distortion; and (III) are practical.

4) We consider the generalization of the HM problem, which removes the requirement that any two #'s in X are at least k positions apart. This problem has a direct application on missing value replacement, where the input set of sensitive patterns corresponds to patterns that are less likely than expected to occur. We adapt the algorithms of Sections 6 and 7 to address this problem. In particular, we show that our methods substantially outperform missing value strategies employed by state-of-the-art DNA data processing tools. These results answer how difficult is to apply frequent pattern mining and missing value replacement from the upper bound point of view. See Section 9.

A preliminary version of this paper appeared in [35].

2 Related Work

Our work is related to three areas: (I) data sanitization (a.k.a. knowledge hiding) [36], [37], which aims to prevent the mining of confidential knowledge from a disseminated dataset, (II) anonymization [38], which aims to prevent the inference of information about individuals represented in a

1. 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 $\mathcal{O}(f(p) \cdot P(q))$ [34].

disseminated dataset, and (III) missing value treatment [39]. We next briefly review related works in these areas.

2.1 Data Sanitization

Data sanitization approaches are typically applied to a collection of transactions [5], [18], [40], a collection of sequences [6], [19], [20], or a single sequence [41]. These approaches employ integer programming [18], [40], dynamic programming [41], or heuristics [5], [6], [19], [20]. The objective of these approaches is twofold: to reduce the frequency (support) of sensitive patterns, so that they cannot be mined at a given frequency threshold τ; and to preserve data utility, often by preserving the set of frequent patterns that can be mined at threshold τ [5], [18], [19], [20], [40]. The patterns considered in these approaches are: itemsets in [5], [18], [40], subsequences in [6], [19], [20], and single letters in [41].

We discuss approaches for sanitizing a collection of sequences in more detail. The works of [6], [19], [20] considered the general problem of hiding a given set of sensitive patterns from an input collection of sequences, so that no sensitive pattern occurs as a *subsequence* (and not as a substring) in at least τ sequences in the collection. To deal with the problem, they proposed deletion-based [6], [19] or permutation-based [20] heuristics. The work of [41] considered the problem of hiding a given set of sensitive events (i.e., single letters) from an event sequence, in which each event is a multi-set of letters that is associated with a timestamp. To deal with the problem, it proposed a dynamic programming algorithm.

Unlike our work, all the approaches that were discussed so far do not aim at hiding sensitive strings, nor at minimizing changes to the set of frequent substrings.

As discussed in Introduction, in the recently proposed approaches for string sanitization [8], [16], [17], #'s must be ultimately replaced so that the locations of sensitive patterns are not exposed. To this end, [8] considered the problem of replacing #'s so as to minimize the total cost of τ-ghost occurrences and showed that this problem is NP-hard. Note that HM, the problem of minimizing the *total number* of τ-ghosts we consider here, is fundamentally different from the problem of minimizing the *total cost* of τ-ghost occurrences and, in particular, it cannot be reduced from Multiple-Choice Knapsack because no arbitrary weights or costs are involved. On the hardness side, this makes our hardness proof considerably more challenging. On the algorithmic side, [8] proposed a heuristic inspired by algorithms for Multiple-Choice Knapsack. This heuristic assumes that each # replacement forces *all* length-k strings that *could* become τ-ghosts with this replacement, to *actually do* become τ-ghosts. Based on this assumption, it assigns a cost to every replacement of every #, and then chooses the replacements that minimize the total cost of τ-ghost occurrences. Due to this pessimistic assumption, this heuristic may not be effective at minimizing the number of τ-ghosts.

2.2 Data Anonymization

Data anonymization approaches for string data are applied to a collection of strings [7], [42], [43], [44], [45], [46], or to a single string [47], [48], [49].

We first discuss approaches applied to a collection of strings. Some works [7], [42], [43] propose heuristics, based on k-anonymity [50]. The goal of [7], [42] is to create a synthetic string that represents a cluster of strings in the input dataset, while that of [43] is to upper-bound the probability of inferring any letter in any string of the published collection of strings. Other works [44], [45], [46] propose heuristics based on differential privacy [51]. The goal of [45] is to release a differentially private string collection. On the other hand, [44] and [46] focus on frequent substrings: [44] aims to release differentially private top-k frequent substrings, where k denotes the number of frequent substrings required, while [46] aims to release differentially private frequent substrings with gap constraints [52].

We now discuss approaches applied to a single string [47], [48], [49]. The goal of [47] is to prevent inferences about a given set of sensitive sequences, by limiting the mutual information between the frequency distribution of sensitive sequences in the string before and after anonymization. To achieve this, it proposed heuristics which replace letters with other letters that represent more abstract (coarse) information. The goal of [48] is to prevent sensitive sequences from occurring within a time window of a temporally-annotated string. To achieve this, it proposed algorithms that delete letters from a string, while preserving occurrences of non-sensitive sequences. The goal of [49] is to prevent the inference of the exact frequency (multiplicity) of any length-k substring in a string based on differential privacy. To achieve this, it proposed exact polynomial-time algorithms based on dynamic programming and linear programming, as well as several linear-time heuristics.

We stress that the above data anonymization approaches are not alternatives to our approach. This is because they cannot be applied to hide a collection of sensitive patterns while preserving utility for frequent pattern mining.

2.3 Missing Value Treatment

Missing values occur in string datasets for a number of reasons [39], and they need to be treated to improve the quality of obtained statistics [53], query answers [25], and data mining models (e.g., association rules [54], [55], sequential patterns [26], clustering [56], and classification [57]). Therefore, existing works remove [53] or replace missing values [25], [56], [57], or alternatively utilize interestingness measures that are suited to mining patterns with missing values [26], [54]. Hence these works are tailored to specific settings and cannot deal with our problem. This is because they do not aim at minimizing the impact that replacing missing values in a string has on frequent pattern mining.

3 Preliminaries and Problem Statement

An *alphabet* Σ 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 Σ. We fix a *string* $X = X[0] \cdots X[n-1]$ of length $|X| = n$ over $\Sigma_{\#}$. The set of length-k strings over Σ is denoted by Σ^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 X. $\mathrm{Freq}_X(U)$ 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 Σ is a set of strings over Σ. We will consider a dictionary of length-k strings that do not occur in X, referred to as *sensitive patterns*. Any element of Σ^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., [58]).

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 $\mathbf{S} \subseteq \Sigma^k$ such that no $S \in \mathbf{S}$ 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. These strings are called τ-*ghosts*.
II No sensitive pattern $S \in \mathbf{S}$ occurs in Z.

Note that function g replaces each $\#$ by exactly one letter from Σ. Condition $|X_i| \geq k-1$, for all $i \in [0,\delta]$, 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 [8, Lemma 1] produces an X satisfying this condition, for *any* given set $\mathbf{S}$, to guarantee that the frequency of every non-sensitive pattern is preserved in X. Thus, HM is directly applicable to the output of [8]. We also consider the generalized version of the HM problem, in which we drop the condition $|X_i| \geq k-1$, for all $i \in [0,\delta]$, specifying that $\#$ occurrences must not be close to each other. This problem is referred to as GENERALIZED HIDE & MINE (GHM).

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 τ-ghost pattern occurs in Z.
II No sensitive pattern $S \in \mathbf{S}$ occurs in Z.

4 HMD IS NP-COMPLETE

Problem HMD is clearly in NP, as the presence of τ-ghosts or sensitive patterns can be verified in polynomial time. In this section, we show that HMD is strongly NP-complete via exhibiting a reduction from a variant of the Bin Packing problem [33]. As a consequence, HM is NP-hard. In what follows, we will denote an instance of a problem P with $\mathcal{I}_\mathrm{P}$.

4.1 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), as well as a vector $[w_1,\ldots,w_N]$ of positive integers representing the weights of the items, the BP problem asks whether we can partition the items into M subsets (bins) without exceeding the capacity of any bin. Formally, we need to decide whether there exists a function $f : [N] \to [M]$, assigning items to bins, such that:

$$\forall i \in [M], \sum_{j\in[N], f(j)=i} w_j \leq B.$$

Crucially, BP is *strongly* NP-complete [33], i.e., it is NP-complete even when weights and bin capacities are bounded by a polynomial function of N and M. In the following, we will consider this case, and use gadgets whose size is proportional to the numerical values in $\mathcal{I}_\mathrm{BP}$, as if we were representing those numbers in unary notation.

We will assume that no two items have the same weight. We refer to this variant of BP as the UNIQUE-WEIGHTS BIN PACKING (UWBP) problem; see Supplemental Material for an example. To justify the unique weights assumption, we show that UWBP is still strongly NP-complete by a polynomial-time reduction from standard BP.

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

Proof. Consider an instance $\mathcal{I}_\mathrm{BP} = M, B, N, w_1, \ldots, w_N$ of BP with possibly duplicated weights, where all values are polynomial in the size of $\mathcal{I}_\mathrm{BP}$: we construct in polynomial time an instance $\mathcal{I}'_\mathrm{BP} = M', B', N', w'_1, \ldots, w'_{N'}$ of UWBP (where no two weights are the same) that has polynomial values, and has a positive answer if and only if $\mathcal{I}_\mathrm{BP}$ does.

To obtain $\mathcal{I}'_\mathrm{BP}$ we proceed as follows. Firstly, set $M' = M$, $N' = N$ and $B' = B \cdot N^2 + (N^2 - 1)$. To obtain the weights w'_i multiply each w_i by N^2, then add "flavoring" by taking groups of items with the same weight one by one and, for each group, adding 0 to its first item, 1 to the second, 2 to the third, and so on. Essentially, we increase the scale of the numbers (a 1-weight item becomes N^2-weight) so much that we can make all weights different without affecting the way groups of items fit in bins: the extra (N^2-1) capacity in B' does not allow to fit an extra unit of item-weight (that is N^2 weight), but it is enough to account for the flavoring of any set of items. Indeed, in the worst case (when all items have the same weight), the cumulative amount of flavoring added to all N items is $0 + 1 + 2 + \ldots + N - 1 < N^2/2$. Hence, an assignment of items to bins is valid for $\mathcal{I}_\mathrm{BP}$ if and only if it is valid for $\mathcal{I}'_\mathrm{BP}$. □

4.2 Overview of the Reduction from UWBP to HMD

We now show that, for any UWBP instance, we can produce in polynomial time an instance of HMD that has positive answer if and only if the UWBP instance 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, with a $\#$ in the middle. We will explain how all UWBP constraints are linked to the gadgets. It will then suffice to concatenate the gadgets into one long string, to obtain an instance of HMD that implies a solution of UWBP.

First, we will consider gadgets t_{ij}, which model whether item j is placed in bin i. The structure of these gadgets ensures that the maximum capacity B of the bins is not exceeded.

Then, gadgets u_{ij} will be introduced. The structure of this second kind of gadgets, together with t_{ij}, ensures that each item is placed in some bin.

The set of sensitive patterns $\mathbf{S}$ and the threshold τ will be carefully chosen to build and link the gadgets. Sensitive patterns will be used to force a specific subset of letters to replace a $\#$ (by forbidding the length-k strings obtained from unwanted replacements).

In essence, to replace a # inside a t_{ij} gadget we will only have two choices: one corresponding to the positive choice "place the j-th item into the i-th bin", and one to the negative choice of not doing so. The first choice will create w_j copies of some length-k substring specific to bin i; the capacity of bin i is modeled by the number of such substrings we can create without exceeding the threshold. On the other hand, the gadgets u_{ij} are there to ensure that, for each item $\hat{\jmath}$, at least one # among the $t_{i\hat{\jmath}}$ is replaced with the positive choice, that is, each item is placed in at least one bin.[2] The threshold τ is essential in linking the gadgets and modeling the capacity of the bins: since no pattern that occurs less than τ times is allowed to reach that same threshold after the replacements, we will repeat the pattern in the string so as to bound the number of its additional occurrences that can be created by replacing a #.

4.3 Construction of an Instance of HMD

The alphabet of the string X of the instance of HMD will be made of letters #, x, y, \$, and a letter b_i for each $i \in [M]$.

For $i \in [M]$, $j \in [N]$, and $k = \max_j w_j + 3$, we define the gadgets t_{ij} and u_{ij} as the following strings of length $2k-1$:

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

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

Example 3. Consider an instance $\mathcal{I}_{\text{BP}}$ with $M = 2$, $B = 5$, $N = 3$, $w_1 = 2, w_2 = 5, w_3 = 3$, which will be the running example for how to build a corresponding instance of HMD along this section. We will have $\Sigma = \{b_1, b_2, \#, x, y, \$\}$ and $k = \max_j w_j + 3 = 8$. The gadgets t_{1j} are:

$$t_{11} = b_1 x x x x x b_1 \# b_1 b_1 b_1 b_1 b_1 b_1 b_1$$
$$t_{12} = b_1 x x b_1 b_1 b_1 b_1 \# b_1 b_1 b_1 b_1 b_1 b_1 b_1$$
$$t_{13} = b_1 x x x x b_1 b_1 \# b_1 b_1 b_1 b_1 b_1 b_1 b_1.$$

Gadgets t_{2j} only differ from gadgets t_{1j} in that in the former b_1 is substituted with b_2. For the same $\mathcal{I}_{\text{BP}}$, gadgets u_{1j} are:

$$u_{11} = b_1 x x x x x b_1 \# y y x x x x y$$
$$u_{12} = b_1 x x b_1 b_1 b_1 b_1 \# y y y y y y y$$
$$u_{13} = b_1 x x x x b_1 b_1 \# y y y x x x y.$$

Again, u_{2j} can be obtained from u_{1j} by replacing b_1 with b_2.

For the sake of readability, from now on we will write U^ℓ to denote $\underbrace{U \dots U}_{\ell}$ (i.e., ℓ concatenations of a string U starting with the empty string). We then define $\mathbf{S}$, the set of sensitive patterns, as 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}.

Example 4. Continuing the running example, the sensitive patterns set for the corresponding instance of HMD will be

$$\mathcal{S} = \{b_1 b_2^7, b_2 b_1^7, b_1 y b_1^6, b_2 y b_2^6, b_1 \$ b_1^6, b_2 \$ b_2^6, b_1 y^2 x^4 y,$$
$$b_1 y^5 x y, b_1 y^3 x^3 y, b_2 y^2 x^4 y, b_2 y^5 x y, b_2 y^3 x^3 y, b_1 \$ y^2 x^4,$$
$$b_1 \$ y^5 x, b_1 \$ y^3 x^3, b_2 \$ y^2 x^4, b_2 \$ y^5 x, b_2 \$ y^3 x^3\}.$$

As explained below, we will use t_{ij} and u_{ij} to construct an instance of string X. By this definition of $\mathbf{S}$, the # in a t_{ij} can only be replaced with b_i or x, and the # in a u_{ij} only with x or y, so that X does not contain sensitive patterns.

We model the size B of the bins using the threshold τ: specifically, we link the filling of the i-th bin with the number of occurrences of a specific non-sensitive pattern (namely, b_i^k). However, this is not the only pattern we need to constrain: we have many different length-k substrings that come into play, all of which need specific thresholds. Thus, a common threshold τ for all non-sensitive patterns is too restrictive. We implement this by choosing τ high enough, and artificially lowering the allowed occurrences of each specific non-sensitive pattern by adding an appropriate amount of extra copies of the non-sensitive pattern itself at the end of the string. This way we can choose a different threshold for each non-sensitive pattern.

In accordance with this reasoning, we choose $\tau = \max\{M, B\} + 1$. We finally construct the string X as a concatenation of the following components, separated by \$\$ as follows:

1) t_{ij}, $\forall i, j$
2) u_{ij}, $\forall i, j$
3) $\tau - B - 1$ occurrences of b_i^k, $\forall i$
4) $\tau - 2$ occurrences of $b_i x^{k-w_j-1} b_i^{w_j-1} x$, $\forall i, j$
5) $\tau - M$ occurrences of $y^{w_j+1} x^{k-w_j-2} y$, $\forall j$.

Component (3) ensures that a valid solution of this instance cannot add more than B occurrences of any b_i^k. Each time we replace the # in a t_{ij} with b_i (corresponding to assigning item j to bin i), we introduce w_j additional occurrences of b_i^k: this models the consumption of space in each bin, and the limit B ensures that no bin overflows.

By Component (4), for each i, j, only one additional occurrence of $b_i x^{k-w_j-1} b_i^{w_j-1} x$ can be created, either by replacing the # with x in a t_{ij} or in a u_{ij}. This ensures that, if we substitute x for # in one of the two gadgets, then we cannot do the same in the other one. Let us consider a specific item j; if we do not place it in bin i, then we are forced to substitute y for # in u_{ij}, creating an occurrence of length-k substring $y^{w_j+1} x^{k-w_j-2} y$. Since, by Component (5), we can only add $M-1$ occurrences of this latter pattern over all M bins, there must be an i such that the # in u_{ij} is replaced with x. The corresponding # in t_{ij} is then forced to be replaced with a b_i, ensuring that item j is assigned to some bin.

2. Note that our reduction technically allows placing an item in several bins, however such a solution can trivially be turned into a proper one by selecting one of the bins arbitrarily and removing the item from all others.

Example 5. To conclude the running example, the instance of HMD equivalent to the original $\mathcal{I}_{\text{BP}}$ is given by the string

$$X = t_{11}\$\$t_{12}\$\$t_{13}\$\$t_{21}\$\$t_{22}\$\$t_{23}\$\$u_{11}\$\$u_{12}\$\$u_{13}\$\$u_{21}\$\$ u_{22}\$\$u_{23}(\$\$b_1x^5b_1x\$\$b_1x^2b_1^4x\$\$b_1x^4b_1^2x\$\$b_2x^5b_2x\$\$ b_2x^2b_2^4x\$\$b_2x^4b_2^2x)^4(\$\$y^3x^4y\$\$y^6xy\$\$y^4x^3y)^4$$

of length $n = 562$ over alphabet $\Sigma_{\#} = \{b_1, b_2, x, y, \$, \#\}$, with $k = 8$, $\tau = \max\{M, B\} + 1 = 6$, and the sensitive patterns set $\mathcal{S}$ given before.

4.4 Correctness

We have shown how to construct in polynomial time an instance $\mathcal{I}_{\text{HMD}}$ from any given instance $\mathcal{I}_{\text{UWBP}}$. We now prove that $\mathcal{I}_{\text{HMD}}$ has a positive answer if and only if $\mathcal{I}_{\text{UWBP}}$ does. For the sake of readability, let us refer to the # in t_{ij} and u_{ij} as $\#^t_{ij}$ and $\#^u_{ij}$, respectively. The solution to $\mathcal{I}_{\text{HMD}}$ can then be expressed via a function $g : \{\#^t_{ij}, \#^u_{ij},\ \forall i, j\} \to \Sigma$. Let $f : [N] \to [M]$ be a solution for a given $\mathcal{I}_{\text{UWBP}}$. We create the corresponding solution to $\mathcal{I}_{\text{HMD}}$ in the following manner, for each item $j \in [N]$ and bin $i \in [M]$:

$$f(j) = i \Rightarrow g(\#^t_{ij}) = b_i \text{ and } g(\#^u_{ij}) = x;$$

$$f(j) \neq i \Rightarrow g(\#^t_{ij}) = x \text{ and } g(\#^u_{ij}) = y.$$

For a given item j such that $f(j) = i$, we get w_j occurrences of b_i^k, one occurrence of $b_i x^{k-w_j-1} b_i^{w_j-1} x$, and for all $h \neq i$ one occurrence of $b_h x^{k-w_j-1} b_h^{w_j-1} x$ and $y^{w_j+1} x^{k-w_j-2} y$. Since the bin capacity in the solution of UWBP is not overflown, we added at most B copies of b_i^k for each i. Finally, since each element is taken once in UWBP, we created exactly $(M-1)$ occurrences of $y^{w_j+1}x^{k-w_j-2}y$ and one occurrence of $b_i x^{k-w_j-1} b_i^{w_j-1} x$. We thus do not create τ-ghosts, and we have a valid solution for HMD.

Vice versa, given a solution g to our HMD instance, to obtain the solution to the original UWBP, it suffices to prove that the following two claims are satisfied:

1) We do not overload any bin; formally
$$\forall i \in [M] \sum_{j \in [N] \text{ s.t. } g(\#^t_{ij}) = b_i} w_j \leq B.$$
2) Each item is assigned to some bin; formally
$$\forall j \in [N] \qquad |\{i \in [M] \text{ s.t. } g(\#^t_{ij}) = b_i\}| \geq 1.$$

If these claims are satisfied, we can extract an assignment for UWBP: for every item j we choose an arbitrary bin i such that $g(\#^t_{ij}) = b_i$, and set $f(j) = i$. By construction of the instance of HMD, these claims are satisfied. By Lemma 1, we obtain the following result.

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

5 HM IS HARD TO APPROXIMATE

Given the hardness of HMD, in this section, we shift our focus on checking whether an approximately optimal solution of HM can be obtained instead. Unfortunately, in Theorem 2, we show that there is no approximation algorithm for HM with additive or multiplicative guarantees unless P=NP. This provides necessary justification for developing alternative approaches, which we describe next.

Theorem 2. *There are no $\alpha \geq 1, \beta \geq 0$ such that there is an approximation algorithm A in P which answers HM by γ with $\gamma \leq \alpha \cdot OPT + \beta$, unless P=NP.*

Proof. Assume there are $\alpha \geq 1, \beta \geq 0$ such that there exists such an approximation algorithm A. We could use A to solve HMD: If $\gamma > \beta$, then we know OPT ≥ 1 and the instance cannot be solved without any τ-ghosts. Otherwise we create a new instance with $\beta + 1$ copies of X, separated by k special letters \$, and each copy with a different alphabet $\Sigma_i = \{a_i : a \in \Sigma\}$. Let γ' be the solution of A on this new instance. Either $\gamma' > \beta$ and $\text{OPT}' = (\beta + 1) \cdot \text{OPT} \geq 1$ so OPT > 1, else $\gamma' \leq \beta$, so there is one of the $\beta + 1$ copies that can be solved without any τ-ghost, which gives us a solution with no τ-ghost for the original instance. □

The reader may now wonder whether the problem becomes easier should one relax the requirement for a *fixed threshold* τ. 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 $\mathbf{S} \subseteq \Sigma^k$ such that no $S \in \mathbf{S}$ 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 τ_1-ghost occurs in Z.
II No sensitive pattern $S \in \mathbf{S}$ occurs in Z.

The practical rationale for considering HMMT is that it could be useful if, for instance, τ_1 is only slightly larger than τ in a given HM instance. Unfortunately, we show that HMMT is NP-hard, and it is even hard to approximate. Due to these provably negative results, we conclude that there is no theoretical gain in studying HMMT instead of HM.

Corollary 3. HMMT *is NP-hard.*

Proof. We reduce HMD to HMMT. Let $\mathcal{I}_{\text{HMD}}$ be the instance of HMD we would like to solve for some threshold τ. We construct an instance of HMMT consisting of the X, k, and $\mathbf{S}$ 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 τ_1-ghost. Then $\mathcal{I}_{\text{HMD}}$ has a positive answer if and only if the answer τ_1 of $\mathcal{I}_{\text{HMMT}}$ is equal to $\tau_0 = \tau$. The statement thus follows. □

Observe that a pattern U is a τ-ghost if and only if $\tau \in (\text{Freq}_X(U), \text{Freq}_Z(U)]$. Therefore, the minimal number of τ-ghosts is not monotonous in τ. On the contrary, the minimal number of τ-ghosts is zero when $\tau = 0$ and all patterns are already frequent (i.e., they appear at least τ times), or when $\tau > n$ and the threshold is so high that no pattern can ever become a τ-ghost. In between, the minimal number of τ-ghosts increases whenever τ 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 $\tau_1 > \alpha(\alpha\tau_0 + \beta) + \beta$ (and for which we can prove that

$\tau_1 \notin [\tau_0 + 1, \alpha(\alpha\tau_0 + \beta) + \beta]$), implying both additive and multiplicative inapproximability.

Theorem 4. *There are no $\alpha \geq 1, \beta \geq 0$ such that there is an approximation algorithm A in P which answers* HMMT *by γ with $\alpha^{-1}(OPT - \beta) \leq \gamma \leq \alpha \cdot OPT + \beta$, unless P=NP.*

Proof. Let X be an arbitrary string and $\mathbf{S}$ 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 $\mathbf{S}'$ of sensitive patterns, so that if an (α, β)-approximation algorithm existed for HMMT, we could decide HMD in polynomial time. We define Y over $\Sigma \cup \{\#, \&\}$ to be

$$Y = X(\&\&T)^{\tau_0}\&(\#T\&)^{\lceil(\alpha^2-1)\tau_0+\alpha\beta+\beta\rceil}.$$

Let $\mathcal{R}$ be the set of all strings $\&sT$, with $s \in \Sigma$. We define the dictionary of sensitive patterns be $\mathbf{S}' = \mathbf{S} \cup \mathcal{R}$. Note that we need to replace all $\#$'s in $(\#T\&)^{\lceil(\alpha^2-1)\tau_0+\alpha\beta+\beta\rceil}$ 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 τ_0 to $\lceil\alpha(\alpha\tau_0 + \beta) + \beta\rceil$. Therefore, if $\tau = \tau_0$, then the number of τ-ghosts in Z equals that in $Z(\&\&T)^{\tau_0}\&(\&T\&)^{\lceil(\alpha^2-1)\tau_0+\alpha\beta+\beta\rceil}$, because the additional new patterns were already occurring at least τ times in Y. However if $\tau_0 < \tau \leq \lceil\alpha(\alpha\tau_0+\beta)+\beta\rceil$, then there will always be at least one τ-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 $\tau_1 > \lceil\alpha(\alpha\tau_0 + \beta) + \beta\rceil$. We conclude that there exists no (α, β)-approximation algorithm for HMMT, unless P=NP. □

6 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* (FPT) for several combinations of parameters. We recall that a problem with parameters p and q is fixed-parameter tractable in p if there exists a function f and a polynomial P such that the problem has time complexity $\mathcal{O}(f(p) \cdot P(q))$ [34].

6.1 ILPs for HMD and HM

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 δ occurrences of $\#$ in X, and that any two occurrences are at least k letters apart, so UV is in Σ^{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 τ-ghost, i.e., if an additional occurrence of N can be created by replacing some $\#$ by a letter in Σ and $\mathrm{Freq}_X(N) \in [\tau - k\delta, \tau - 1]$. 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 τ for N to become τ-ghost. We assign to each critical string N a unique id ℓ, and denote it by N_ℓ. We introduce the following parameters:

$\boldsymbol{\gamma}$	number of distinct contexts present in X;
$\boldsymbol{\delta_i}$	number of occurrences of $\#_i$ 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_ℓ introduced by replacing a $\#_i$ with $j \in \Sigma$, 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_ℓ: the number of its additional occurrences we can afford. Since replacing an occurrence of $\#_i$ by $j \in \Sigma$ adds k new strings in Σ^k, $\boldsymbol{\alpha^i_{\ell,j}}$ counts how many of them are equal to N_ℓ. Let $\boldsymbol{x_{i,j}}$ be the number of times we replace $\#_i$ by $j \in \Sigma$, and $\mathcal{F} \subseteq [\gamma] \times \Sigma$ 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 τ-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 τ-ghosts.

Let us now focus on solving HM. As opposed to HMD, we can decide in polynomial time if HM has a solution: we check all $|\Sigma|$ letter replacements at each of the δ 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, for the rest of this paper, we assume that HM always has a solution. To minimize τ-ghosts in $Z = X_0 g(1) X_1 g(2) \cdots g(\delta) X_\delta$, that is, the number of strings $U \in \Sigma^k$ with $\mathrm{Freq}_X(U) < \tau$ and $\mathrm{Freq}_Z(U) \geq \tau$, we define a binary variable z_ℓ, $\ell \in [\lambda]$, which is equal to 1 when N_ℓ has become τ-ghost, and is equal to 0 otherwise. 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

$$\begin{cases} x_{i,j} \geq 0 & \forall (i,j) \in [\gamma] \times \Sigma \\ x_{i,j} = 0 & \forall (i,j) \in \mathcal{F} \\ z_\ell \geq 0 & \forall \ell \in [\lambda] \\ \sum_{i\in[\gamma],j\in\Sigma} \alpha^i_{\ell,j} x_{i,j} - k\delta z_\ell \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j\in\Sigma} x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{2}$$

Note that, in the ILP of Eq. 2, if N_ℓ is a τ-ghost then $\sum_{i\in[\gamma],j\in\Sigma} \alpha^i_{\ell,j} x_{i,j} - k\delta z_\ell \leq e_\ell$ if and only if $z_\ell = 1$.

6.2 HMD and HM are FPT

Eq. 1 is clearly an ILP with $m = \gamma|\Sigma|$ variables and at most $2m + \lambda + \gamma$ constraints. The algorithm by Frank and Tardos [59] 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 $\mathbf{S}$. The value (id) of each key (letter) is chosen from $\{1, \ldots, k|\mathbf{S}| + n\}$. This construction can be done in $\mathcal{O}(k|\mathbf{S}| + n)$ time using perfect hashing [60]. We can then lexicographically sort all length-k substrings of X and all length-k strings in $\mathbf{S}$ (viewed as strings over letter id's) using radix sort in $\mathcal{O}(k|\mathbf{S}| + kn)$ time, and construct two dictionaries, one for X and one for $\mathbf{S}$, as follows. The dictionary for X is a trie of all its non-sensitive length-k substrings, in which each such substring is associated to its frequency in X. The dictionary for $\mathbf{S}$ is simply a trie of all its strings. In both tries, for every node, we store the first letter on each of its outgoing edges in a static dictionary with $\mathcal{O}(1)$ access time [60]. 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 $\mathcal{O}(k|\mathbf{S}| + kn)$.

Observe that $\delta = \mathcal{O}(n/k)$. When $\delta = \mathcal{O}(1)$, the brute-force algorithm checking all possible ways to replace the #'s with letters of Σ runs in polynomial time. There are indeed $|\Sigma|^\delta$ ways to replace the #'s; each way generates δk new length-k strings for which we must check if they are sensitive or create a τ-ghost. We can check if they are sensitive using the trie of $\mathbf{S}$ in $\mathcal{O}(k)$ time per each such string. We can count the additional number of occurrences of each length-k substring of X using the trie of X in $\mathcal{O}(k)$ time. Finally, we can count the number of occurrences of each length-k string not occurring in X by constructing a trie of all (at most δk) such strings, similar to the preprocessing step. This gives $\mathcal{O}(kn + k|\mathbf{S}| + k^2\delta|\Sigma|^\delta)$ time in total.

The following theorems explain when an FPT algorithm exists for HMD and for HM.

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_ℓ for these strings) in $\mathcal{O}(kn)$ time. The id i of the context of each # and its number δ_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, all values $\alpha^i_{\ell,j}$ and set $\mathcal{F}$ can be computed in $\mathcal{O}(\gamma|\Sigma|k^2)$ total time as follows. When we replace $\#_i$ with a letter j we create k new length-k strings, 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). We check if they are sensitive using the trie of $\mathbf{S}$ in $\mathcal{O}(k)$ time per string; we count the additional number of occurrences of a critical length-k substring of X using the trie of X in $\mathcal{O}(k)$ time; we finally count 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) by constructing a trie of all such strings, similar to the preprocessing step. The ILP is thus constructed in $\mathcal{O}(kn + k|\mathbf{S}| + \gamma|\Sigma|k^2)$ 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 [59], HMD is FPT if γ and $|\Sigma|$ are fixed.

(b) Since every context has length $2k - 2$ and both $|\Sigma|$ and k are $\mathcal{O}(1)$, 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, and therefore we cannot directly solve the ILP in polynomial time. However, note that the only letters we need to discern are the ones contained in the λ critical length-k strings, which are at most λk in total. Since we do not need to distinguish between the rest of the letters, we can represent all of them using the same special letter. Let $\sigma \subseteq \Sigma$ denote the set of letters contained in critical length-k strings, which can be determined as described in (a): σ can be specified and indexed using perfect hashing [60] 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:

$$\begin{cases} x_{i,j} \geq 0 & \forall i \in [\gamma], j \in \sigma \cup \{\$\} \\ x_{i,j} = 0 & \forall (i, j) \in \mathcal{F}|_\$ \\ \sum_{i \in [\gamma], j \in \sigma} \alpha^i_{\ell,j} x_{i,j} \leq e_\ell & \forall \ell \in [\lambda] \\ \sum_{j \in \sigma \cup \{\$\}} x_{i,j} = \delta_i & \forall i \in [\gamma] \end{cases} \tag{3}$$

This new ILP can be constructed in $\mathcal{O}(kn + k|\mathbf{S}| + \gamma|\Sigma|k^2)$ time, like Eq. 1. Since the ILP has only $\gamma(|\sigma| + 1) = \mathcal{O}(1)$ variables, HMD is FPT for fixed k and λ [59]. 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. □

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. 2 in $\mathcal{O}(\lambda)$ time from the ILP of Eq. 1, which can be constructed in $\mathcal{O}(kn + k|\mathbf{S}| + \gamma|\Sigma|k^2)$ time; see the proof of Theorem 5(a). The ILP of Eq. 2 has at most $2m + 2\lambda + \gamma$ constraints and $m + \lambda = |\Sigma|\gamma + \lambda$ variables. Therefore HM is FPT if $|\Sigma|$, γ and λ are fixed [59].

(b) Similar to the ILP of Eq. 3 (see Theorem 5(c)), we can reduce the alphabet Σ 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 λ are fixed [59]. □

Theorems 5 and 6 show that we are able to design polynomial-time algorithms for HMD and HM when some input parameters to these problems are fixed, despite the hardness of the problems. This is particularly encouraging because these parameters are small in most real cases.

7 Greedy Heuristic for HM

We present a heuristic that aims at minimizing τ-ghosts by controlling the number and frequency of length-k strings that may become τ-ghosts. Our heuristic performs two left-to-right passes over the input string X to incrementally construct Z from left to right. In the first pass, it computes statistics by creating a dictionary $\mathcal{T}_S$ that stores all sensitive

patterns in $\mathbf{S}$ as strings of length k. This dictionary can be implemented using a hash table or a trie, and supports membership queries for $\mathbf{S}$. Moreover, our heuristic creates a (hash or trie) map $\mathcal{T}_{X\cup Z}$ that stores pairs $(Y, \mathrm{Freq}(Y))$, where the key Y is a length-k substring that either appears in X or is created when an occurrence of $\#$ is replaced by a letter in Z. The associated value $\mathrm{Freq}(Y)$ is given by the number of occurrences of Y in X (possibly zero) plus any new occurrences in the current Z created by $\#$ replacements. The reason for using $\mathcal{T}_{X\cup Z}$ rather than just the occurrences of Y in Z, is to get better statistics by knowing "some future" (i.e., the remaining part of X in which $\#$ are yet to be expanded but some occurrences of Y may be found). The query supported for a length-k string Y is the following: if any pair $(Y, \mathrm{Freq}(Y))$ exists, it is unique and the value of $\mathrm{Freq}(Y)$ is returned; otherwise, the value zero is returned. Initially $\mathcal{T}_{X\cup Z}$ stores the statistics for X alone, as Z has yet to be generated. As discussed next, the construction of Z is incrementally performed from left to right in the second pass, where our heuristic greedily replaces the occurrences of $\#$, based on the statistics maintained using $\mathcal{T}_{X\cup Z}$ and in a way that aims at minimizing τ-ghosts.

The pseudocode of our heuristic is provided in Algorithm 1. In the first pass (Lines 1 to 3), the heuristic constructs $\mathcal{T}_S$ and $\mathcal{T}_{X\cup Z}$ to efficiently maintain pattern frequencies, and also initializes Z, which maintains the sanitized string. Then, the heuristic performs the second pass over X in Lines 4 to 23, scanning some letters of X from left to right. If the current letter $X[i] \neq \#$, then it is simply appended to Z in Line 23. Therefore, we focus on the main case, when $X[i] = \#$: the heuristic considers the context UV and iterates over each letter j in the alphabet $\Sigma\cup\{\epsilon\}$ to find a replacement j^* (if any[3]) for $X[i]$ as follows (Lines 7 to 21). It constructs the set S_j of all length-k substrings of string $U\cdot j\cdot V$ (Line 11). If S_j contains no sensitive patterns (i.e., $S_j\cap\mathcal{T}_S=\emptyset$), the heuristic considers the subset $S_j^{<\tau}\subseteq S_j$ containing those length-k substrings with frequency less than τ, and computes $\sum_{Y\in S_j^{<\tau}}(\tau-\mathrm{Freq}(Y))^{-1}$ through queries to the map $\mathcal{T}_{X\cup Z}$ (Lines 12 to 16). Thus, it computes a *gap measure* indicating how far from τ are the frequencies of the potential patterns that may become τ-ghosts in $S_j^{<\tau}$. If j^* is empty then the heuristic fails, as all replacements of $\#$ with a letter $j\in\Sigma\cup\{\epsilon\}$ would reinstate a sensitive pattern (Line 18). Otherwise, the heuristic replaces $\#$ with j^* as the latter optimizes the gap measure, it appends both j^* and V to Z, and increases the frequencies in the map $\mathcal{T}_{X\cup Z}$ with the frequencies of the strings in S_{j^*} as substrings of $U\cdot j^*\cdot V$ (Lines 19 to 21). After completing the second pass over X, the heuristic returns Z and terminates (Line 24).

An example of our heuristic is in Supplemental Material.

Our heuristic takes $\mathcal{O}(k|\mathbf{S}|+kn+\delta|\Sigma|k^2)=\mathcal{O}(k|\mathbf{S}|+kn|\Sigma|)$ time as $\delta=\mathcal{O}(n/k)$. The first two terms in $\mathcal{O}(k|\mathbf{S}|+kn+\delta|\Sigma|k^2)$ correspond to the cost of constructing $\mathcal{T}_S$ and $\mathcal{T}_{X\cup Z}$. The third term is the cost of the second left-to-right pass. As can be seen in Lines 5 to 21, this is dominated by the cost of processing each of the δ occurrences of $\#$ in X, which requires $\mathcal{O}(|\Sigma|k^2)$ time, as processing S_j takes $\mathcal{O}(k^2)$ time for a letter $j\in\Sigma\cup\{\epsilon\}$, plus the $\mathcal{O}(n)$-time scan of X, which is in turn dominated by the term $\mathcal{O}(|\Sigma|k^2)=\mathcal{O}(kn|\Sigma|)$.

3. Following [8], we added the empty letter ϵ to Σ to model the deletion of $\#$'s as this can lower the number of τ-ghosts.

Algorithm 1 GREEDY-HEURISTIC$(k, X, \Sigma, \mathbf{S}, \tau)$

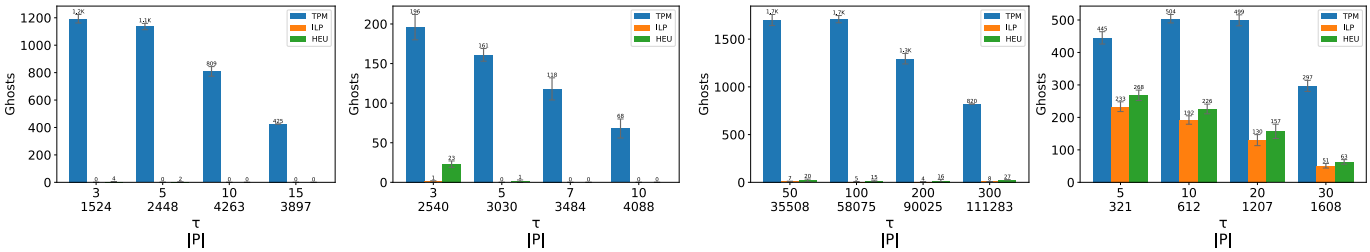
Require: wlog X has no $\#$ in its first and last $k-1$ positions

1: $\mathcal{T}_S \leftarrow$ dictionary storing all sensitive patterns in $\mathbf{S}$
2: $Z \leftarrow X[0\,..\,k-2]$
3: $\mathcal{T}_{X\cup Z} \leftarrow$ map storing pairs $(Y, \mathrm{Freq}(Y))$ for all non-sensitive length-k substrings of X plus those added in Z
4: $i \leftarrow k-1$
5: **while** $i < |X|$ **do**
6: **if** $X[i] = \#$ **then**
7: $U \leftarrow Z[\,|Z|-k+1\,..\,|Z|-1\,]$
8: $V \leftarrow X[\,i+1\,..\,i+k-1\,]$
9: $best \leftarrow +\infty$
10: **for** each letter $j \in \Sigma\cup\{\epsilon\}$ **do**
11: $S_j \leftarrow$ set of length-k substrings of $U\cdot j\cdot V$
12: **if** $S_j \cap \mathcal{T}_S = \emptyset$ **then**
13: $sum \leftarrow 0$
14: $S_j^{<\tau} \leftarrow \{Y \in S_j \mid \mathrm{Freq}(Y) < \tau\}$
15: **for** each string $Y \in S_j^{<\tau}$, using $\mathcal{T}_{X\cup Z}$ **do**
16: $sum \leftarrow sum + (\tau - \mathrm{Freq}(Y))^{-1}$
17: **if** $sum < best$ **then** $j^* \leftarrow j$; $best \leftarrow sum$
18: **if** $best = +\infty$ **then return** FAIL
19: $Z \leftarrow Z\cdot j^*\cdot V$
20: Update $\mathcal{T}_{X\cup Z}$ for the strings in S_{j^*}
21: $i \leftarrow i+1+|V|$
22: **else**
23: $Z \leftarrow Z\cdot X[i]$; $i \leftarrow i+1$ *// no update of* $\mathcal{T}_{X\cup Z}$
24: **return** Z

There are three benefits of the heuristic compared to the exact algorithm: (I) It has polynomial time complexity, even when none of the input parameters of HM is fixed. (II) It can be trivially adapted to address the GHM problem within the same time complexity (see Section 9). (III) By design, it prevents large increases in the frequency of patterns that do not become τ-ghosts but have increased frequency as a result of $\#$ replacement, since the sum computed in Lines 14–16 increases with $\mathrm{Freq}(Y)$. This helps reducing distortion, a secondary consideration in sanitization [5], [8], [16], [18], [19], [20]. We have also considered two variants of the heuristic (at no extra time cost), which replace the sum $\sum_{Y\in S_j^{<\tau}}(\tau-\mathrm{Freq}(Y))^{-1}$ computed in Lines 14–16 with $|S_j^{<\tau}|$; or with $\max_{Y\in S_j^{<\tau}}(\tau-\mathrm{Freq}(Y))^{-1}$. Clearly, the former aims at minimizing the number of patterns that could become τ-ghosts by subsequent letter replacements without considering their frequency, while the latter aims at reducing the frequency of the substring that is closer to become τ-ghost by subsequent letter replacements.

8 EXPERIMENTS

Experimental Setup and Datasets. The string sanitization method of [8] takes as input a string W over Σ, a positive integer k, and a set $\mathbf{S}$ of sensitive patterns, and then it performs the following three steps: (I) It constructs the shortest string X over $\Sigma_\#$ such that X contains no sensitive pattern and the order (and thus frequency) of all non-sensitive patterns in X and W is the same (see Section 1). (II) It further tries to minimize the length of X by preserving the exact frequency of non-sensitive patterns but relaxing



(a) OLD (b) TRU (c) MSN (d) DNA

Fig. 1: Number of τ-ghosts for each dataset and varying τ (on the top of each bar, we show the number of τ-ghosts). The values of $|P|$ are averaged over 10 runs.

the order property, so that instead of a total order a partial order is preserved. The output of this step is a string Y over $\Sigma_{\#}$. (III) It replaces $\#$'s in Y by the Multiple-Choice Knapsack based heuristic (see Section 2). The output of this step is a string Z over Σ.

In our evaluation, we performed Steps (I) and (II) to obtain Y, which we process by different methods: the ILP formulation in Eq. 2 (denoted by ILP), our greedy heuristic (denoted by HEU), or the heuristic of [8] described in Step (III) (denoted by TPM). Following [8], we added the empty letter ϵ to the set of letters that may be used to replace $\#$. This effectively models the deletion of $\#$'s and can lower the number of τ-ghosts. We omit the results for the other variants of our heuristic because HEU outperformed them.

The utility of any sanitized string Z is measured by two well-established utility measures for sanitized data:

1) The number of τ-ghosts in Z; i.e., the size of the set $\{U \in \Sigma^k : \text{Freq}_X(U) < \tau \text{ and } \text{Freq}_Z(U) \geq \tau\}$. All tested methods are guaranteed to create no τ-*lost*, i.e., the set $\{U \in \Sigma^k : \text{Freq}_X(U) \geq \tau \text{ and } \text{Freq}_Z(U) < \tau\}$ is empty. Clearly, zero τ-lost and τ-ghost patterns imply no utility loss for frequent length-k substring mining.
2) The *Distortion* measure [8], which is defined as $\sum_U (\text{Freq}_W(U) - \text{Freq}_Z(U))^2$, where $U \in \Sigma^k$ is a non-sensitive pattern. This measure penalizes changes in the frequency of non-sensitive patterns; low values imply high utility for frequency-based tasks [61].

Minimizing the number of τ-ghosts is crucial, as it is the primary goal of data sanitization [5], [18], [19], [20]. Distortion considers the frequency of *all* patterns and can thus be seen as a secondary criterion aiming to capture utility when the sanitized dataset is released for frequency-based tasks other than frequent pattern mining.

We used publicly available datasets that were also used in the evaluation of [8]: Oldenburg (OLD) [62], Trucks (TRU) [63], MSNBC (MSN) [64], and the complete genome of *Escherichia coli* (DNA) [65]. OLD contains movement data, TRU contains transportation data, MSN contains clickstream data, and DNA contains genomic data. We also used uniformly random string datasets, referred to as SYN1 and SYN2. See Table 1a for the characteristics of these datasets. In this table, an interval for a parameter contains the values we used for that parameter. Let us remark that $|\mathbf{S}|$ denotes the number of sensitive patterns whereas $|P|$ denotes the total number of positions where a sensitive pattern occurs in the input string.

TABLE 1: (a) Dataset characteristics. (b) Default values used.

Dataset	length n	alphabet size $\lvert\Sigma\rvert$	num sens patterns $\lvert\mathbf{S}\rvert$	num sens positions $\lvert\mathcal{P}\rvert$	pattern length k	threshold τ
OLD	85,563	100	[60, 480]	[606, 6663]	[3, 7]	[3, 15]
TRU	5,763	100	[40, 160]	[920, 4137]	[2, 5]	[5, 30]
MSN	4,698,764	17	[100, 600]	[30036, 180118]	[4, 10]	[50, 300]
DNA	4,641,652	4	[30, 60]	[321, 1608]	[9, 15]	[5, 30]
SYN1	20,000,000	10	[10, 1000]	[1994, 2000537]	[3, 6]	[5, 20]
SYN2	5,000,000	10	[10, 1000]	[79, 3500168]	[3, 6]	[5, 15]

(a)

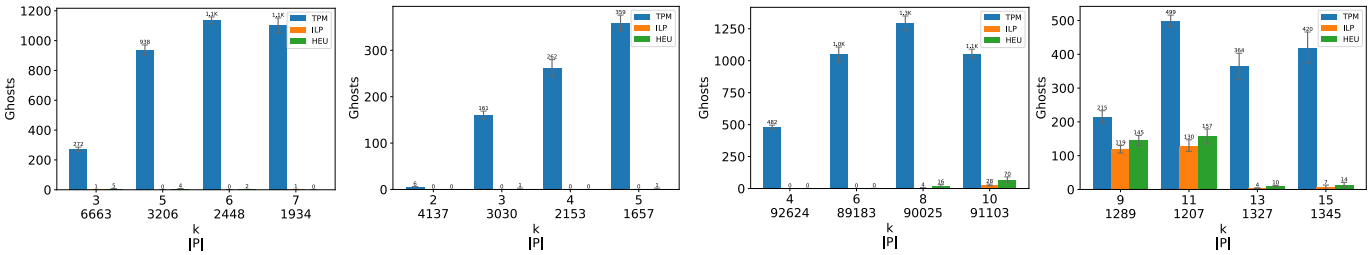
Dataset	num sens patterns $\lvert\mathbf{S}\rvert$	pattern length k	threshold τ
OLD	240	6	5
TRU	120	3	5
MSN	300	8	200
DNA	50	11	20
SYN1	100	5	10
SYN2	700	6	7

(b)

The configuration of parameters was performed as in [8] (see Table 1b for default values). That is, the sensitive patterns were selected randomly among the frequent length-k substrings of minimum support τ, following [8], [16]. This is because there are no patterns that are known to be sensitive in these datasets. We averaged the results over 10 runs, following [6]. The weight, costs, and θ parameters in TPM were configured as in [8]. Our code was written in C++ and is available at https://github.com/fnareoh/hide_and_mine. The code of TPM was also written in C++ and is available at [66]. We used the Gurobi solver v. 9.0.1 (single-thread configuration) to solve ILP instances. All experiments ran on an Intel Core i7-10810U CPU @ 1.10 GHz with 32GB RAM, which indicates the low computational requirements of the methods.

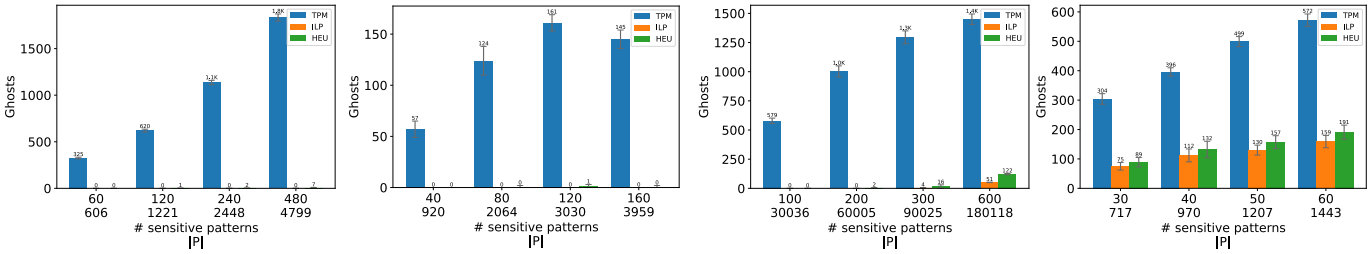
Data Utility. We show that our methods: (I) allow for frequent length-k pattern mining with no or negligible utility loss (i.e., they create zero or few τ-ghosts), unlike TPM, and (II) incur substantially lower distortion than TPM.

Number of τ-ghosts. We examined the impact of τ, k, and $|\mathbf{S}|$ on τ-ghosts, in Figs. 1, 2, and 3, respectively. As can be seen, ILP and HEU created a significantly smaller number of τ-ghosts than TPM in all cases. This is because ILP finds the best possible solution by design, while HEU specifically tries to avoid the creation of τ-ghosts by limiting the frequency of critical patterns. TPM did not perform well because it does not explicitly consider the number of τ-ghosts; it only tries to minimize the total cost of τ-ghost occurrences, as discussed in Section 2. Note that DNA was challenging to sanitize with few τ-ghosts, because its small alphabet size makes it difficult to find letters that



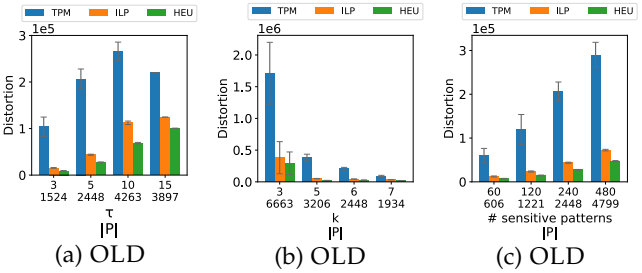
(a) OLD (b) TRU (c) MSN (d) DNA

Fig. 2: Number of τ-ghosts for each dataset and varying pattern length k (on the top of each bar, we show the number of τ-ghosts). The values of $|P|$ are averaged over 10 runs.



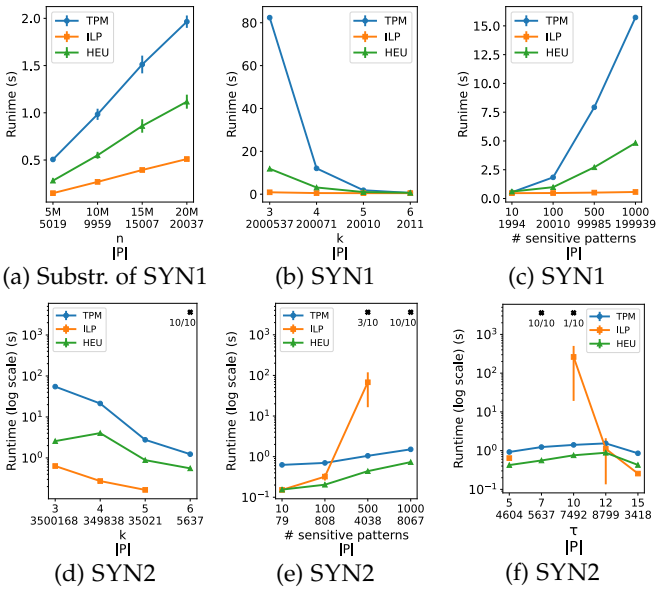
(a) OLD (b) TRU (c) MSN (d) DNA

Fig. 3: Number of τ-ghosts for each dataset and varying number of sensitive patterns $|\mathbf{S}|$ (on the top of each bar, we show the number of τ-ghosts). The values of $|P|$ are averaged over 10 runs.



(a) OLD (b) OLD (c) OLD

Fig. 4: Distortion for varying: (a) τ, (b) k, and (c) number of sensitive patterns $|\mathbf{S}|$. The values of $|P|$ are averaged over 10 runs.



(a) Substr. of SYN1 (b) SYN1 (c) SYN1

(d) SYN2 (e) SYN2 (f) SYN2

Fig. 5: Runtime for varying: (a) n, (b) k, and (c) $|\mathbf{S}|$ on SYN1. Runtime for varying: (d) k, (e) $|\mathbf{S}|$, and (f) τ on SYN2. The values of $|P|$ are averaged over 10 runs. An ✖ with ratio $x/10$ corresponds to an instance in which ILP was stopped after 1 hour to produce a *feasible solution* in x out of 10 runs.

replace #'s without creating τ-ghosts. Again, on DNA, ILP and HEU outperformed TPM by 1548% and 795% on average, while HEU was worse than ILP by 137% on average, which is expected as ILP guarantees minimizing τ-ghosts. Our results show that both our methods allow for substantially more accurate frequent pattern mining than TPM and indicate that the heuristic which replaces #'s in TPM is ineffective to minimize the number of τ-ghosts.

Distortion. We examined the impact of τ, k and $|\mathbf{S}|$ on Distortion, in Fig. 4. Our methods outperformed TPM because its objective function favors the replacements of #'s with letters that increase the frequency of already frequent patterns. Increasing the frequency of such patterns does not incur τ-ghosts, but significantly increases Distortion (see the Distortion computation formula). HEU outperformed ILP, incurring 37% lower Distortion on average on the OLD dataset for the reason mentioned in Section 7. Further discussion of this reason and additional results are in Supplemental Material.

Runtime. We examined the impact of input string length n, k, τ, and $|\mathbf{S}|$ on runtime. We considered two different settings using SYN1 and SYN2, respectively.

Setting I. As can be seen in Fig. 5a, our methods required less than one second to process the 20-million letter string. They also remained efficient when sanitizing patterns of different length (Fig. 5b), as well as when sanitizing a large number of sensitive patterns (Fig. 5c). An interesting observation from Fig. 5c is that ILP scaled much better than

TPM and HEU with respect to $|\mathbf{S}|$. This is because it groups #'s by context when formulating the last two constraints in Eq. 2 in terms of γ (i.e., in terms of the number of distinct contexts). Thus, its runtime may improve when a larger number of #'s have the same context due to the grouping. This is different from TPM and HEU which replace #'s one by one and thus spend time for computing the alternative ways in which each # can be substituted. For example, the time spent by HEU for this process is $\mathcal{O}(|\Sigma|k^2)$ per #, as can be seen from Lines 7 to 21. Overall, ILP was the fastest and HEU was substantially faster than TPM. We omit the runtime experiments for varying τ because they were quantitatively similar to those reported here.

Setting II. As can be seen in Figs. 5d, 5e, and 5f, HEU again outperformed TPM substantially in all cases. However, ILP was not consistently faster than HEU and TPM as in Setting I. In some cases, it took less than 1 second to produce an optimal solution, while in others it did not produce an optimal solution within 1 hour. In the latter cases, we stopped the solver after 1 hour to obtain a feasible (suboptimal) solution. As expected, these cases correspond to instances in which a very large number of τ-ghosts could be incurred. This can be seen in Supplementary Material, along with experiments for varying n. Overall, the experiments in this setting establish the main benefit of HEU over ILP: predictable running time due to its guaranteed polynomial-time complexity.

Similar observations can be made in the case of real datasets. For example, ILP was faster than HEU in the case of DNA, whereas HEU was faster in the case of OLD (e.g., HEU was 2.5 times faster, taking 0.4 seconds on average over the results of Fig. 1a). Furthermore, there were cases when ILP took too much time. For example, in the experiment of Fig. 3c, ILP took 36 minutes to run on MSN when the number of sensitive patterns was 600, while HEU took only seconds. Also, in the experiment of Fig. 1b, ILP took 20 seconds to run on TRU when τ was 3, while HEU took less than 0.12 seconds.

9 The Generalized Hide & Mine Problem

The Generalized Hide & Mine (GHM) problem is the generalized version of HM, in which we drop the condition specifying that occurrences of # must not be close to each other. In particular, any two occurrences of # are not necessarily at least k positions apart. Since HM is NP-hard and hard to approximate, it follows that the more general GHM is also NP-hard and hard to approximate. However, GHM cannot be addressed by our exact algorithm for HM. This is because it does not consider how combinations of #'s can give rise to sensitive and non-sensitive pattern occurrences. To address this issue, we proceed as follows.

We consider all occurrences of #'s in X: for each such occurrence $s \in [\delta]$, we consider all occurrences $t \in [\delta]$ within k positions to the right of occurrence s. Let $\mathcal{P}$ be the set of all such pairs (s,t) for all s. For each pair $(s,t) \in \mathcal{P}$ and each array over Σ of possible replacements $J = (J_s, \ldots, J_t)$, we check which patterns are created by the substitution of the i-th occurrence of # by J_i, for all $i \in [s,t]$ (note, we omit patterns which are created by substituting a proper prefix or suffix of these #'s). If this substitution creates a sensitive pattern occurrence, we set $(s,t,J) \in \mathcal{F}$. Otherwise let $\alpha^{s,t}_{\ell,J}$ be the number of occurrences of N_ℓ it creates (recall from Section 6 that N_ℓ is a critical string). Moreover, let $y_{(s,t),J} \in \{0,1\}$ be such that (I) if each occurrence of # from s to t is replaced by its corresponding letter from J, then $y_{(s,t),J} = 1$, and (II) if replacing each occurrence of # from s to t introduces some sensitive pattern, then $y_{(s,t),J} = 0$. In any other case $y_{(s,t),J}$ can have either value. Variable $x_{i,j} \in \{0,1\}$ simply indicates whether the i-th occurrence of # is replaced by letter $j \in \Sigma$, in contrast with Section 6, where $x_{i,j} \in \mathbb{Z}^{\geq 0}$ accounted for the number of times an occurrence of # with context i was replaced by $j \in \Sigma$.

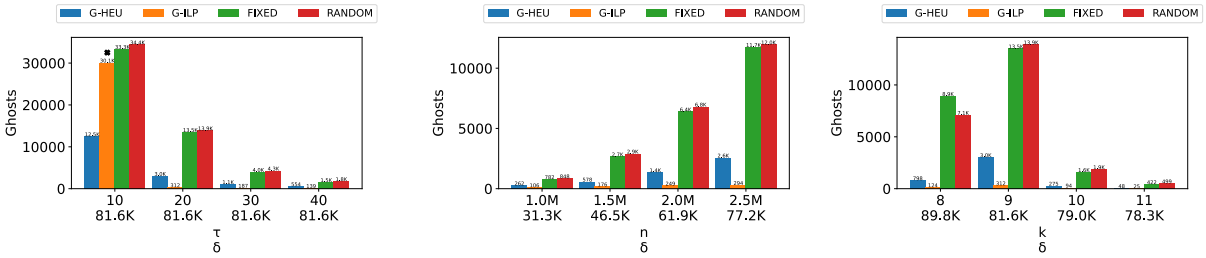
The other variables are defined as in Section 6. The ILP formulation for GHM is to find $x \in \{0,1\}^{\delta \times |\Sigma|}$ and $y_{(s,t),J} \in \{0,1\}$ for all $\{(s,t,J) \mid (s,t) \in \mathcal{P}, J \in \Sigma^{t-s+1}\}$ so as to minimize $\sum_{\ell=1}^{\lambda} z_\ell$ subject to

$$\begin{cases} 0 \leq x_{i,j} \leq 1 & \forall (i,j) \in [\delta] \times \Sigma \\ 0 \leq y_{(s,t),J} \leq 1 & \forall (s,t) \in \mathcal{P}, J \in \Sigma^{t-s+1} \\ y_{(s,t),J} = 0 & \forall (s,t,J) \in \mathcal{F} \\ z_\ell \geq 0 & \forall \ell \in [\lambda] \\ t - s + y_{(s,t),J} \geq \sum_{i \in [s,t]} x_{i,J_i} & \forall (s,t) \in \mathcal{P}, J \in \Sigma^{t-s+1} \\ \sum_{j \in \Sigma} x_{i,j} = 1 & \forall i \in [\delta] \\ \sum_{\substack{(s,t) \in \mathcal{P}, \\ J \in \Sigma^{t-s+1}}} \alpha^{s,t}_{\ell,J} y_{(s,t),J} - k\delta z_\ell \leq e_\ell & \forall \ell \in [\lambda] \end{cases} \tag{4}$$

Constraints 1 to 4, 6 and 7 of Eq. 4 are analogous to constraints 1 to 5 in Eq. 2 of Section 6. Constraint 5 of Eq. 4 states that an array of #'s is replaced by an array of letters, if each of those #'s is replaced by the corresponding letter.

Note that, since for any $t - s + 1$ #'s occurring within k positions we have variables for all $J \in \Sigma^{t-s+1}$, the size of this ILP grows exponentially in the number of #'s that can be in a pattern. Even if we remove variables $y_{(s,t),J}$ and the corresponding constraints 2 and 5 for all substitutions J that do not create any critical or sensitive patterns, the number of constraints and variables of the ILP can grow exponentially, as there can be exponentially many critical patterns (and thus variables $\alpha^{s,t}_{\ell,J}$). This ILP can therefore be of exponential size even when the set of sensitive patterns is empty. To construct a feasible solution in polynomial time, we can slightly modify the heuristic in Section 7: looking at Algorithm 1, the only required modification is in Line 8, where V is now assigned the substring $X[i+1\,..\,i+\ell]$ for the largest $\ell < k$ such that $X[i+1\,..\,i+\ell]$ does not contain any #'s; V can be empty as any two #'s can be consecutive.

We demonstrate the impact of GHM in the context of missing value replacement. A missing value corresponds to a #, and the set of sensitive patterns to patterns that are much less likely than expected to occur, based on *deviation*, a well-established statistical significance measure for strings [30], [31] (see Supplemental Material). These patterns would be an artefact of missing value replacement, and thus we do not allow them to occur in the output string. We evaluate the ILP formulation in Eq. 4 and our modified heuristic, denoted by G-ILP and G-HEU, respectively, against two alternative strategies: (I) FIXED and (II) RANDOM. FIXED replaces every occurrence of # with the same letter, which is selected from $\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$. RANDOM replaces every occurrence of # with a letter selected uniformly at random from $\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$. These strategies are employed by state-of-the-art DNA data processing tools (e.g., [28], [29]).



(a) READS (b) READS (c) READS

Fig. 6: Number of τ-ghosts for varying: (a) τ, (b) input string length n, and (c) k. The ✖ symbol in (a) corresponds to an instance in which G-ILP was stopped after 1 hour to produce a *feasible solution*. (δ is the total number of #'s.)

We applied G-HEU to READS, a real dataset comprised of $75,446$ mouse reads each of length 35 and containing at least one missing value (unknown DNA base represented by #) [67]. The dataset has total length $n = 2,640,610$, it contains $77,107$ #'s, and the alphabet size is $|\Sigma| = 4$, with $\# \notin \Sigma$. The default values were $k = 9$ and $\tau = 20$; as sensitive patterns we used the 100 patterns that were the least likely than expected to occur, according to deviation. All experiments ran on the PC mentioned in Section 8.

We measured the number of τ-ghosts incurred by all methods in Fig. 6. Specifically, Figs. 6a, 6b, and 6c show the impact of τ, n, and k on the number of τ-ghosts, respectively. Our methods outperformed FIXED and RANDOM, which shows their effectiveness at minimizing τ-ghosts. As can be seen in Fig. 6a, a larger τ leads all methods to create fewer τ-ghosts because there are fewer frequent patterns and thus fewer of them may become τ-ghosts. As in Section 8, G-ILP did not produce an optimal solution within 1 hour when there was a large number of τ-ghosts that could be incurred (see also Supplemental Material), while the other methods finished within seconds. In this case, we stopped G-ILP after 1 hour to obtain a feasible (suboptimal) solution. As can be seen in Fig. 6b, a larger n leads all methods to create more τ-ghosts, because there are more #'s to replace. Also, observe in Fig. 6c that the number of τ-ghosts for all methods increases from $k = 8$ to $k = 9$ and then decreases as k gets larger. The increase for $k = 8$ and $k = 9$ is because there are more frequent patterns compared to when k is larger and hence more patterns may become τ-ghosts. The decrease for $k = 10$ and $k = 11$ is because there are more total possible length-k patterns (their number grows exponentially in k), so it is easier to create distinct length-k patterns and avoid τ-ghosts. Overall, ILP performed much better than HEU, but in difficult instances it did not finish within 1 hour, while both ILP and HEU vastly outperformed FIXED and RANDOM.

10 Outlook

In addition to strings, frequent pattern mining is also applied on other data types, such as graphs, trees, itemsets etc. [12]. Given the fact that string is one of the most basic data types, our hardness results support the intuition that replacing missing values with no utility loss for frequent pattern mining in these more complex data types may not be possible in polynomial time; based on our results, we further anticipate that it might even be hard to approximate such solutions in polynomial time. Given the successful deployment of ILP in the string representations presented in this paper, ILP might be a promising strategy to be applied for replacing missing values in other data representations and settings. Also, it is interesting to design ILP formulations that consider additional utility requirements, such as preserving the segmental structure of the input string [68] or the frequency of certain substrings.

Acknowledgments

GB is partially supported by the Netherlands Organisation for Scientific Research (NWO) under project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)" AC, RG, and NP are partially supported by the University of Pisa under the "PRA – Progetti di Ricerca di Ateneo" (Institutional Research Grants) - Project no. PRA_20202021_26 "Metodi Informatici Integrati per la Biomedica". GG is partially funded by the grant ANR-20-CE48-0001 from the French National Research Agency (ANR). GL is partially supported by the Leverhulme Trust RPG-2019-399 project. NP, SPP, and LS are partially supported by the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 956229. SPP and LS is partially supported by the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 872539. MS and LS are supported by the Netherlands Organisation for Scientific Research (NWO) through Gravitation-grant NETWORKS-024.002.003.

References

[1] J. J. Ying, W. Lee, T. Weng, and V. S. Tseng, "Semantic trajectory mining for location prediction," in *SIGSPATIAL*, 2011, pp. 34–43.
[2] R. Agrawal and R. Srikant, "Mining sequential patterns," in *ICDE*, 1995, pp. 3–14.
[3] D. C. Koboldt, K. M. Steinberg, D. E. Larson, R. K. Wilson, and E. R. Mardis, "The next-generation sequencing revolution and its impact on genomics," *Cell*, vol. 155, no. 1, pp. 27–38, 2013.
[4] M. Chen, X. Yu, and Y. Liu, "Mining moving patterns for predicting next location," *Inf. Syst.*, vol. 54, no. C, pp. 156–168, 2015.
[5] Y. Wu, C. Chiang, and A. L. P. Chen, "Hiding sensitive association rules with limited side effects," *TKDE*, vol. 19, no. 1, pp. 29–42, 2007.
[6] O. Abul, F. Bonchi, and F. Giannotti, "Hiding sequential and spatiotemporal patterns," *TKDE*, vol. 22, no. 12, pp. 1709–1723, 2010.

[7] C. C. Aggarwal and P. S. Yu, "A framework for condensation-based anonymization of string data," *DMKD*, vol. 16, no. 3, pp. 251–275, 2008.

[8] 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*, 2019, pp. 627–644.

[9] H. M. Martinez, "An efficient method for finding repeats in molecular sequences," *Nucleic Acids Res.*, vol. 11, no. 13, pp. 4629–4634, 1983.

[10] U. Keich and P. A. Pevzner, "Finding motifs in the twilight zone," *Bioinform.*, vol. 18, no. 10, pp. 1374–1381, 2002.

[11] L. Marsan and M. Sagot, "Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification," *J. Comput. Biol.*, vol. 7, no. 3-4, pp. 345–362, 2000.

[12] W. Shen, J. Wang, and J. Han, "Sequential pattern mining," in *Frequent Pattern Mining*, 2014, pp. 261–282.

[13] H. Arimura and T. Uno, "An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence," *J. Comb. Optim.*, vol. 13, no. 3, pp. 243–262, 2007.

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

[15] I. Ajunwa, K. Crawford, and J. Ford, "Health and big data: An ethical framework for health information collection by corporate wellness programs," *J. of Law, Medicine and Ethics*, vol. 44, pp. 474–480, 2016.

[16] G. Bernardini, H. Chen, A. Conte, R. Grossi, G. Loukides, N. Pisanti, S. P. Pissis, G. Rosone, and M. Sweering, "Combinatorial algorithms for string sanitization," *TKDD*, vol. 15, no. 1, pp. 8:1–8:34, 2021.

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

[18] A. Gkoulalas-Divanis and V. S. Verykios, "Exact knowledge hiding through database extension," *TKDE*, vol. 21, no. 5, pp. 699–713, 2009.

[19] A. Gkoulalas-Divanis and G. Loukides, "Revisiting sequential pattern hiding to enhance utility," in *KDD*, 2011, pp. 1316–1324.

[20] R. Gwadera, A. Gkoulalas-Divanis, and G. Loukides, "Permutation-based sequential pattern hiding," in *ICDM*, 2013, pp. 241–250.

[21] V. Guralnik and G. Karypis, "A scalable algorithm for clustering sequential data," in *ICDM*, 2001, pp. 179–186.

[22] 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," *Bioinform.*, vol. 34, no. 7, pp. 1125–1131, 2017.

[23] M. Spiliopoulou, "Managing interesting rules in sequence mining," in *PKDD*, 1999, pp. 554–560.

[24] IUPAC-IUB Commission on Biochemical Nomenclature, "Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents," *Biochemistry*, vol. 9, no. 20, pp. 4022–4027, 1970.

[25] F. Biessmann, D. Salinas, S. Schelter, P. Schmidt, and D. Lange, ""Deep" learning for missing value imputation in tables with non-numerical data," in *CIKM*, 2018, pp. 2017–2025.

[26] C. Fiot, A. Laurent, and M. Teisseire, "Approximate sequential patterns for incomplete sequence database mining," in *FUZZ*, 2007, pp. 1–6.

[27] E. Bier, R. Chow, P. Golle, T. H. King, and J. Staddon, "The rules of redaction: Identify, protect, review (and repeat)," *IEEE Secur. Priv.*, vol. 7, no. 6, pp. 46–53, 2009.

[28] R. Li, C. Yu, Y. Li, T. W. Lam, S. Yiu, K. Kristiansen, and J. Wang, "SOAP2: an improved ultrafast tool for short read alignment," *Bioinform.*, vol. 25, no. 15, pp. 1966–1967, 2009.

[29] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrows-wheeler transform," *Bioinform.*, vol. 26, no. 5, pp. 589–595, 2010.

[30] V. Brendel, J. S. Beckmann, and E. N. Trifonov, "Linguistics of nucleotide sequences: Morphology and comparison of vocabularies," *J. of Biomol. Structure and Dynamics*, vol. 4, no. 1, pp. 11–21, 1986.

[31] M. Régnier and M. Vandenbogaert, "Comparison of statistical significance criteria," *J. Bioinform. and Comput. Biology*, vol. 4, no. 2, pp. 537–552, 2006.

[32] 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*, 2001, pp. 378–385.

[33] M. R. Garey and D. S. Johnson, ""Strong" NP-completeness results: Motivation, examples, and implications," *J. ACM*, vol. 25, no. 3, pp. 499–508, 1978.

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

[35] G. Bernardini, A. Conte, G. Gourdel, R. Grossi, G. Loukides, N. Pisanti, S. P. Pissis, G. Punzi, L. Stougie, and M. Sweering, "Hide and mine in strings: Hardness and algorithms," in *ICDM*, 2020, pp. 924–929.

[36] C. C. Aggarwal and P. S. Yu, *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2008.

[37] F. Bonchi and E. Ferrari, *Privacy-Aware Knowledge Discovery: Novel Applications and New Techniques*. CRC Press, 2010.

[38] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Comput. Surv.*, vol. 42, no. 4, Jun. 2010.

[39] C. C. Aggarwal. and S. Parthasarathy, "Mining massively incomplete data sets by conceptual reconstruction," in *KDD*, 2001, p. 227–232.

[40] E. C. Stavropoulos, V. S. Verykios, and V. Kagklis, "A transversal hypergraph approach for the frequent itemset hiding problem," *Knowl. Inf. Syst.*, vol. 47, no. 3, pp. 625–645, 2016.

[41] G. Loukides and R. Gwadera, "Optimal event sequence sanitization," in *SDM*, 2015, pp. 775–783.

[42] C. C. Aggarwal and P. S. Yu, "On anonymization of string data," in *SDM*, 2007, pp. 419–424.

[43] M. Terrovitis, G. Poulis, N. Mamoulis, and S. Skiadopoulos, "Local suppression and splitting techniques for privacy preserving publication of trajectories," *TKDE*, vol. 29, no. 7, pp. 1466–1479, 2017.

[44] L. Bonomi and L. Xiong, "A two-phase algorithm for mining sequential patterns with differential privacy," in *CIKM*, 2013, pp. 269–278.

[45] R. Chen, G. Acs, and C. Castelluccia, "Differentially private sequential data publication via variable-length n-grams," in *CCS*, 2012, pp. 638–649.

[46] S. Xu, X. Cheng, S. Su, K. Xiao, and L. Xiong, "Differentially private frequent sequence mining," *TKDE*, vol. 28, no. 11, pp. 2910–2926, 2016.

[47] L. Bonomi, L. Fan, and H. Jin, "An information-theoretic approach to individual sequential data sanitization," in *WSDM*, 2016, pp. 337–346.

[48] D. Wang, Y. He, E. Rundensteiner, and J. F. Naughton, "Utility-maximizing event stream suppression," in *SIGMOD*, 2013, pp. 589–600.

[49] H. Chen, C. Dong, L. Fan, G. Loukides, S. P. Pissis, and L. Stougie, "Differentially private string sanitization for frequency-based mining tasks," in *ICDM*, 2021, pp. 41–50.

[50] P. Samarati and L. Sweeney, "Generalizing data to provide anonymity when disclosing information (abstract)," in *PODS*, 1998, p. 188.

[51] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[52] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *EDBT*, 1996, pp. 1–17.

[53] R. J. Little and D. B. Rubin, *Statistical Analysis with Missing Data (3rd ed.)*. USA: John Wiley & Sons, Inc., 2019.

[54] T. Calders, B. Goethals, and M. Mampaey, "Mining itemsets in the presence of missing values," in *SAC*, 2007, pp. 404–408.

[55] A. A. Ragel and B. Crémilleux, "Treatment of missing values for association rules," in *PAKDD*, 1998, pp. 258–270.

[56] J. Tuikkala, L. Elo, O. Nevalainen, and T. Aittokallio, "Missing value imputation improves clustering and interpretation of gene expression microarray data," *BMC Bioinform.*, vol. 9, 2008.

[57] B. Dong, S. Xie, J. Gao, W. Fan, and P. S. Yu, "Onlinecm: Real-time consensus classification with missing values," in *SDM*, 2015, pp. 685–693.

[58] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi, "Text compression using antidictionaries," in *ICALP*, 1999, pp. 261–270.

[59] A. Frank and E. Tardos, "An application of simultaneous diophantine approximation in combinatorial optimization," *Combinatorica*, vol. 7, no. 1, p. 49–65, 1987.

[60] M. L. Fredman, J. Komlós, and E. Szemerédi, "Storing a sparse table with O(1) worst case access time," *J. ACM*, vol. 31, no. 3, pp. 538–544, 1984.

[61] S. P. Pissis, "MoTeX-II: structured MoTif eXtraction from large-scale datasets," *BMC Bioinform.*, vol. 15, p. 235, 2014.
[62] https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm.
[63] https://bitbucket.org/stringsanitization/stringsanitizationpkdd19/src/master/truck_char.txt.
[64] http://archive.ics.uci.edu/ml/datasets/msnbc.com+anonymous+web+data.
[65] http://bacteria.ensembl.org/Escherichia_coli_str_k_12_substr_mg1655/.
[66] https://bitbucket.org/stringsanitization/stringsanitizationpkdd19.
[67] A. Mortazavi, B. Williams, K. McCue, L. Schaeffer, and B. Wold, "Mapping and quantifying mammalian transcriptomes by rna-seq," *Nature Methods*, vol. 5, pp. 621–628, 2008.
[68] G. Shani, C. Meek, and A. Gunawardana, "Hierarchical probabilistic segmentation of discrete events," in *ICDM*, 2009, pp. 974–979.

Giulia Bernardini is an Assistant Professor at the University of Trieste. Her research interests lie in theory of algorithms and their application in bioinformatics and data mining.

Alessio Conte is Assistant Professor at the University of Pisa. His research focuses on efficient subgraph enumeration and mining for "real-world" networks, with applications such as community detection, network design or bioinformatics.

Garance Gourdel is a PhD student at Inria Rennes and ENS. Her topics of interest include algorithms on strings, compact data structures and hashing techniques, as well as their applications to bioinformatics.

Roberto Grossi is a Professor of computer science with the University of Pisa. He has authored or coauthored more than 160 articles in the area of design and analysis of algorithms and data structures.

Grigorios Loukides is an Associate Professor at King's College London. His research interests are in data privacy, data mining, and biomedical informatics.

Nadia Pisanti is an Associate Professor at the University of Pisa. Her research is on algorithms for the analysis of (genomic) data.

Solon P. Pissis is a Senior Researcher at CWI and an Associate Professor at the Vrije Universiteit, both in Amsterdam. His research focuses on theory of algorithms and their application in data mining.

Giulia Punzi is a PhD student at the University of Pisa. Her research focuses on algorithm design and analysis, for problems concerning pattern discovery in strings and graphs.

Leen Stougie is a Senior Researcher at CWI and a Professor of Operations Research at the Vrije Universiteit, both in Amsterdam. He is also a member of the INRIA-Erable team. His research focuses on the design and analysis of algorithms for optimization.

Michelle Sweering is a PhD student at CWI. Her research focuses on combinatorial algorithms on strings and graphs.