Optimal String Sanitization Against Strategic Attackers

Pengxin Bian, *Graduate Student Member, IEEE*, George Theodorakopoulos, Solon P. Pissis, and Grigorios Loukides, *Senior Member, IEEE*

***Abstract*—Strings (sequences of elements) are often disseminated to support applications, e.g., in bioinformatics, web analysis, and transportation. Unfortunately, this may expose sensitive patterns that model confidential knowledge. Concealing the occurrences of sensitive patterns in a string (e.g., by deleting some elements) while minimizing the associated quality loss has been the objective of several string sanitization methods. However, real-world attackers are likely to possess background knowledge about the string, e.g., an individual's genome sequence is almost identical to a reference genome sequence. In addition, it is good security practice to assume that the attacker will know the algorithm that has been used to sanitize the string (Kerckhoffs' principle). Yet, all existing methods fail to protect strings against such attackers, risking privacy breaches in critical applications.**

In our work, we consider for the first time how to defend against *strategic* attackers who possess such knowledge. To achieve this, we propose a novel framework to sanitize a string by probabilistically replacing carefully selected patterns. As part of this framework, we design three mathematical programming algorithms which compute the optimal replacement probabilities under different objectives and constraints, offering different privacy gain / quality loss tradeoffs. Our algorithms protect against strategic attackers using new concepts and measures, protect sensitive patterns of any length, and can construct one or more optimally sanitized strings that can be used in applications such as frequent pattern mining.

Our experiments using five real-world datasets from different domains show that all our algorithms are substantially more effective than a natural baseline (e.g., they offer up to 2 times more privacy when they are configured to incur the same quality loss, and up to 3 times lower quality loss when they are configured to offer the same privacy). They also show that two "hybrid" algorithms that we propose, based on combining elements of the above algorithms, inherit the advantages of their constituent algorithms. These results, coupled with the generality of our approach, make our algorithms practical and beneficial for deployment.

***Index Terms*—Data sanitization, strings, sensitive patterns.**

I. Introduction

A LARGE number of applications ranging from bioinformatics to web analytics and transportation feature *strings* defined as sequences of elements, called *letters*, over an alphabet. For example, in bioinformatics, a DNA sequence can be modeled as a string with each letter of the alphabet $\{\mathtt{A}, \mathtt{T}, \mathtt{C}, \mathtt{G}\}$ representing a DNA base [1], [2]; in web analytics, a user's browsing history can be modeled as a string with each letter representing a visited webpage [3], [4]; and in transportation, a user's location profile can be modeled as a string with each letter representing a visited location [5]–[7].

In all these applications, the strings may contain sensitive (confidential) information. For example, in bioinformatics, some parts of a DNA sequence are linked to diseases [1]; in web analytics, some parts of a user's browsing history may correspond to webpages revealing political beliefs or membership of certain minority groups [8]; and in transportation, some parts of a user's location profile may correspond to locations such as health clinics or temples that are associated with health issues or religious beliefs [9].

Protecting sensitive information in strings is therefore necessary to preserve privacy. This is the goal of *string sanitization* [10]–[17], which aims to protect a given set of *sensitive patterns* modeling the sensitive information by concealing them from the string. To conceal the sensitive patterns, existing string sanitization methods reduce their frequency in the sanitized string (e.g., to zero [10]–[13]) by deleting [14], [16], [17] or permuting [15] some letters from the sensitive patterns, or replacing them with carefully selected nonsensitive patterns [10]–[14]. At the same time, string sanitization methods [11], [12], [14], [15] aim to preserve the set of *nonsensitive τ-frequent* patterns (i.e., nonsensitive patterns that have frequency at least τ before sanitization should have frequency at least τ after sanitization).

Motivation. An important limitation of all existing string sanitization methods is that they assume attackers who possess *no background knowledge* about the input string W or about the algorithm used to sanitize W. They assume that an attacker knows only the sanitized string Z produced from W, the alphabet Σ of W, and the set S of sensitive patterns [10]–[12]. The attacker then succeeds if they can learn that any sensitive pattern occurs in W. Similar assumptions are made by sanitization methods for many other data types (e.g., the methods in [18]–[20] for transaction data, and the method in [16] for trajectory data).

However, attackers often possess background knowledge about the occurrence of certain patterns in W and about the

P. Bian and G. Loukides are with Department of Informatics, King's College London, 30 Aldwych, London WC2B 4BG, UK.
E-mail: {pengxin.bian, grigorios.loukides}@kcl.ac.uk
G. Theodorakopoulos is with Cardiff University, UK. Email: theodorakopoulosg@cardiff.ac.uk
S. P. Pissis is with CWI and the Vrije Universiteit, The Netherlands. Email: solon.pissis@cwi.nl
Manuscript received April 19, 2005; revised August 26, 2015.

sanitization algorithm, which may allow them to break the privacy that string sanitization algorithms offer. Knowledge about such patterns in W can often be obtained from external data sources [21]. For example, a person's DNA sequence is about 99.6% identical to a reference human genome sequence [22]. Thus, when W is a human DNA sequence, the reference sequence, denoted by the string $\mathcal{R}$, may allow an attacker to estimate the probability with which a pattern occurs in W as the probability that the pattern occurs in $\mathcal{R}$. Also, string sanitization algorithms are detailed in published works. Thus, it is reasonable to assume that an attacker knows how these algorithms work, as is standard (Kerckhoffs' principle [23]) in security research. We illustrate this limitation of existing methods using the algorithm of [11] as an example.

Example 1. *Consider the DNA sequence W below and the set of sensitive patterns $S = \{\texttt{ACA}, \texttt{CAC}\}$ which are also highlighted in W. The algorithm of [11] is applied to W to produce the string Z below. The algorithm replaces a sensitive pattern s of length $|s|$ with a concatenation of three strings: The leftmost $|s|-1$ letters of the sensitive pattern, a single letter from the alphabet (chosen in a specific way based on a knapsack-like algorithm; see [11] for details), and the rightmost $|s|-1$ letters of the sensitive pattern. In this example, the sensitive pattern $s =$ ACA of length $|s| = 3$ is replaced by ACCCA, which is the concatenation of AC (leftmost 2 letters of ACA), C (single letter), CA (rightmost 2 letters of ACA). Similarly, CAC is replaced with CAAAC.*

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
W	**T**	A	C	A	**G**	A	A	**T**	A	**G**	C	A	C	T				
Z	**T**	A	C	C	C	A	**G**	A	A	**T**	A	**G**	C	A	A	A	C	T

Consider a strategic attacker who uses a reference sequence $\mathcal{R}$ to learn that only the strings ACA *and* A *(underlined in W above) can occur right after* **T** *and right before* **G***, each with probability* 0.5*. Thus, when the attacker observes the substring* ACCCA *spanning positions 1-5 of Z, right after* **T** *(position 0) and right before* **G** *(position 6), they know that* ACCCA *must occur in Z as a result of sanitization. The attacker also knows that the algorithm of [11] was used to produce Z, and thus they can infer that* ACCCA *has been created by sanitizing* ACA*. Therefore, the attacker infers that the sensitive pattern* ACA *occurs at position* 1 *of W. Similarly, they infer that the sensitive pattern* CAC *occurs at position* 10 *of W.*

Contributions. To address the above limitation, we consider attackers who possess background knowledge in addition to the knowledge assumed by existing methods. Specifically, these attackers know (I) the probability that each substring occurs in certain parts of W and (II) the sanitization algorithm used to construct Z, and they will try to use their knowledge to minimize privacy. We refer to such attackers as *strategic*, in line with the works of [21], [24], [25], which however are applied in a different setting and have a different goal (see Section II for details).

We also propose an algorithmic framework that replaces each sensitive pattern with another string with some probability. The probabilities are computed by any of the optimal algorithms we propose. These algorithms offer different tradeoffs between the privacy gain and the quality loss caused by the replacements. The novelty of our algorithms lies in that they: (I) protect from strategic attackers using new concepts and measures, (II) protect sensitive patterns of any length, (III) provide immediate guarantees of optimality and exploit efficient solvers, as they are based on mathematical programming, and (IV) can construct multiple optimal sanitized strings.

Our specific contributions are as follows.

1. We propose an algorithmic framework to protect sensitive patterns against a strategic attacker. First, we construct the attacker's background knowledge. This involves computing the probability that a pattern occurs in a certain part of W efficiently using a pattern matching algorithm and text indexing. It also involves selecting similar patterns to a sensitive pattern as its potential replacements to reduce quality loss. Second, we compute the optimal probability of replacing each pattern with another pattern, subject to quality loss or privacy gain criteria, using our algorithms, to be described later. Third, we construct the sanitized string Z based on the probabilities computed by the second step.

2. We propose three *optimal* algorithms based on mathematical programming to compute the replacement probabilities, which are used to sanitize strings against strategic attackers. They are optimal in that no algorithm can possibly perform better against an attacker with the same strategic background knowledge of the substring probabilities and of the sanitization algorithm. The algorithms take as input a *context* of a sensitive pattern u (i.e., a pair of strings of a given length, one of which appears right before u and the other appears right after u), a set U of patterns that appear in between the strings of the context, and the attacker's background knowledge. They output the probability of replacing a pattern $u \in U$ with a pattern $u' \in U$, for every pair (u, u'), where u and u' are not necessarily distinct.

The first algorithm is based on linear programming. It maximizes the privacy gain achieved by selecting u' as a replacement averaged over all possible u', subject to a constraint that *upper-bounds the average quality loss* caused by selecting u' as a replacement, over all possible pairs (u, u'). The second algorithm is based on mixed-integer linear programming. It minimizes the average quality loss over all possible pairs (u, u'), subject to a constraint which *lower-bounds the privacy gain* achieved by replacing u with u', for each pair (u, u'). The third algorithm is based on linear programming. It has the same objective function as the second one, but enforces differential privacy [26], a well-known privacy principle.

Next, in Example 2 continuing from Example 1, we illustrate the privacy benefit of our algorithms compared to existing sanitization algorithms.

Example 2 (Cont'd from Example 1)**.** *Consider the string W and the set S of sensitive patterns from Example 1. For the sensitive pattern* ACA*, we first construct the background*

knowledge from W. This involves identifying that ACA *occurs right after* **T** *and right before* **G** *in W with a probability of 0.5 (since* A *also occurs right after* **T** *and right before* **G**). *We then choose the two most similar patterns to* ACA*, namely* A *and* ACA *itself, as potential replacements for* ACA*. Second, we apply our first algorithm, which computes the probability of replacing* ACA *by* A *as 0.9, and that of replacing* ACA *by itself as 0.1. Next, based on these probabilities, we replace the sensitive pattern* ACA *by* A *at position 1 of Z. Lastly, by a similar process, we replace the sensitive pattern* CAC *by* AA *at position 8 of Z. Other Z's are also possible based on the replacement probabilities.*

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
W	**T**	**A**	**C**	**A**	**G**	A	A	**T**	**A**	**G**	C	A	C	T
Z	**T**	A	**G**	A	A	**T**	A	**G**	A	A	T			

The strategic attacker from Example 1 learns from a reference sequence $\mathcal{R}$ that only the strings ACA *and* A *can occur right after* **T** *and right before* **G**, *each with probability 0.5. The attacker may notice that there are some replacements of the sensitive pattern* ACA *right after* **T** *and right before* **G** *by observing Z, because the occurrence probabilities of* A *(respectively, of* ACA*) right after* **T** *and right before* **G** *in Z are 1 (respectively, 0), which do not match the probabilities of 0.5 based on their knowledge.*

However, unlike in Example 1, the attacker cannot learn the positions of the sensitive pattern ACA *in W. This is because each of the replacements of this pattern can occur at each position right after* **T** *and right before* **G** *with the probabilities known to the attacker. Similarly, the attacker cannot learn the positions of* CAC*, as the replacements of this pattern can occur in each position right after* **G** *and right before* **T** *with the probabilities known to the attacker.*

3. We conduct an extensive experimental study on five real-world datasets from different domains, including bioinformatics, transportation, the web, and IoT [27]–[31], using seven quality loss and privacy measures. Since existing sanitization algorithms are not comparable to ours (see Section II), we compare our algorithms to a natural baseline inspired by [21]. Our experiments show that our algorithms outperform the baseline, offering (I) at least 45% and up to more than 2 times higher privacy, when they are configured to incur the same quality loss with the baseline; or (II) at least 77% and up to more than 3 times lower quality loss, when they are configured to offer the same privacy. We also show that our algorithms are practical and their parameters can be configured to trade off privacy and quality.

Finally, we design and evaluate two "hybrid" optimal algorithms; one which adds the constraint of the second algorithm in Contribution 2 to the first, and another which adds the constraint of the third algorithm in Contribution 2 to the second one. Both hybrid algorithms offer a better privacy-quality tradeoff compared to the ones in Contribution 2.

Section II presents related work, Section III our algorithms, and Section IV our experimental evaluation. We conclude the paper in Section V.

II. Related Work

There are two main directions in privacy-preserving data publishing: data sanitization and anonymization. In the following, we discuss data sanitization approaches and refer to Section I of the Supplemental Material for a discussion of data anonymization approaches. The latter are not alternatives to our approach, as they do not aim to conceal a set of sensitive patterns from a string, but rather to prevent the inference of information about individuals from a dataset containing the information of these individuals [32]. In this section, we review methodological papers and refer the reader to [33]–[41] for applications of privacy to specific areas: biomedicine [33], [38], [39], IoT [34], urban mobility [35]–[37], and natural language processing [40], [41].

Data sanitization approaches can be categorized based on the type of information they protect into those that protect: (I) sets of items [19], [20], [42], [43], [44] or rules (relations between sets of items) [18], [45], [46]; (II) trajectories [16]; (III) sequences (elements of a string that are not necessarily consecutive) [14]–[16], [47]; (IV) single elements of a string [17], [21], [24], [25], [48]; and (V) substrings (consecutive elements of a string) [10]–[13], [49].

The works that protect sets of items employ integer programming [19], [20], or heuristics [42], [43], [44]. The works that protect rules employ heuristics [18], [45], [46]; the heuristic of [18] is based on decision tree construction, that of [45] on decreasing the support or confidence of the sensitive rules, and that of [46] on swarm optimization. The work of [16] is applied to a collection of trajectories (i.e., a collection of sequences of spatiotemporal points). It proposes a heuristic algorithm that protects sensitive trajectory patterns in a collection of trajectories by deleting certain elements from them in a greedy fashion. The objective is to reduce the frequency of the sensitive trajectory patterns in the sanitized dataset to at most a given threshold. All the aforementioned works differ from ours in the type of data they consider (i.e., they do not consider a string), the operations they use to conceal sensitive patterns (i.e., they do not replace substrings), and in that they consider attackers with no background knowledge (i.e., they do not consider attackers with knowledge of the sanitization algorithm or of the statistics of the input data).

The works that protect sequences are applied to a collection of strings [14]–[16], or a single string [47]. The former works aim to reduce the frequency of the sequences that they protect in the sanitized collection to at most a given threshold. This ensures that sensitive sequences cannot be mined at this threshold by an attacker with no background knowledge about W. To do this, [14], [16] delete some elements from the sensitive sequences using greedy heuristics aiming to minimize the number of deleted elements in order to preserve data utility [16], or to preserve the set of frequent nonsensitive sequences in the collection [14]. The work of [15] shares the same utility goal as that of [14] but proposes a heuristic that employs permutation instead of deletion. The work of [47] aims to delete a minimal number

Ref.	Algorithm input	Attacker knowledge					Attacker objective
		OS	SP	SA	RLF	RSS	
[14]–[16]	collection of strings	✓	✓				find frequent sensitive patterns
[47]	single string	✓	✓				find frequent sensitive patterns
[21], [24], [25]	single string	✓		✓	✓		find sensitive string elements
[17], [48]	single string	✓	✓				find sensitive string elements
[10]–[13], [49]	single string	✓	✓				find location of sensitive patterns
Our approach	single string	✓	✓	✓		✓	find location of sensitive patterns

TABLE I: Differences among the string sanitization approaches. OS: output string(s); SP: sensitive patterns; SA: sanitization algorithm; RLF: reference letter frequencies; RSS: reference string statistics.

of letters from a string, so as to reduce the frequency of sensitive sequences to zero, and it proposes three heuristic algorithms for this. The works of [14]–[16], [47] differ from ours in the type of data they protect (i.e., they protect sequences), in the operations they use to conceal sensitive patterns (i.e., they do not replace substrings), and in that they consider attackers with no background knowledge (i.e., they do not consider attackers with knowledge of the sanitization algorithm or of the statistics of the input data).

The works that protect single elements of a string protect each element independently of all other elements. The work of [24] formalizes the *localization* attack (i.e., computing the probability distribution over regions where the user might be at a certain time), by considering a *strategic* attacker who has background knowledge of the user's mobility (statistics of the input string), as well as of the protection algorithm. The works of [21], [25] propose approaches to defend against this strategic attacker. They formalize the attacks as a zero-sum Bayesian Stackelberg game and compute strategies for the user and the attacker that are mutually best responses. These approaches provide robust protection for any location inference attack [25]. The works of [17] and [48] consider attackers with no background knowledge; the former works by optimally deleting elements from the string to reduce their frequency in prefixes of the string, and the latter by replacing elements with synthesized ones that have similar semantic features to the actual ones. Our approach differs from [17], [21], [24], [25], [48] in the type of data it protects (a string vs. one string element) and in the background knowledge assumed by the attacker (reference string statistics vs. reference letter frequencies [21], [24], [25], or no background knowledge [17], [48]).

The works that protect substrings [10]–[13], [49] are the most relevant to our work. All of these works consider sensitive substrings that have the same length k and reduce their frequency in the sanitized string to zero. To achieve this, they add nonsensitive parts of the string and/or new letters, including potentially a special letter $\#$ that is not contained in the alphabet of the input string. These works differ in their utility objectives. The work of [12] aims to construct a sanitized string with a minimum length that also preserves the order of the length-k nonsensitive substrings in the input string. The works of [11] and [13] aim to construct a sanitized string that preserves the order of all length-k nonsensitive substrings but is at minimal edit distance from the original string. The work of [49] aims to construct a sanitized string that is at minimal k-gram distance from the original string. A limitation of [11]–[13], [49] is that they may introduce spurious patterns that cannot be mined from the original string, but can be mined from the sanitized string [10], [11]. Since such spurious patterns harm data utility, the work of [10] studies the problem of minimizing their number, proposing integer linear programming algorithms that replace all $\#$'s. The novelty of our approach compared to [10]–[13], [49] is that it considers stronger attackers, it is applicable to sensitive substrings of different length (not only fixed-length ones), it uses different data transformation strategies (probabilistic pattern replacement), and it has different utility objectives (see Section I).

A summary of the differences among the string sanitization approaches is in Table I.

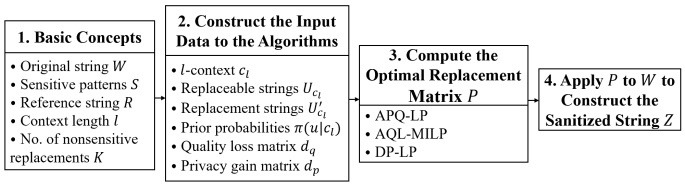


Fig. 1: Flow chart of our algorithmic framework.

III. Algorithmic Framework

In this section, we discuss our sanitization framework. Central to our framework is a reference string, parts of which are known to the attacker and referred to as contexts (see Section III-A). The probability that a substring of the reference string is associated to a context is computed for selected strings, and it is given as input to our sanitization algorithms along with matrices quantifying the quality loss and privacy gain from sanitization (see Section III-B). The attacker model, our sanitization algorithms, and the construction of the sanitized string are discussed in Section III-C, III-D, and III-E, respectively. See Fig. 1 for a flow chart.

A. Basic Concepts

An alphabet Σ is a finite nonempty set of elements called *letters*. A *string* X is a sequence of letters over Σ. The concatenation of two strings X, Y is denoted by $X \cdot Y$. For two positions i and j on X, $X[i] \cdot \ldots \cdot X[j]$ is the *substring* of X that starts at position i of X and ends at position j. We also say that this substring *occurs* at position i. A

substring Y of X can occur multiple times in X. The set of its occurrences (starting positions) is denoted by $\mathtt{occ}_X(Y)$.

By W we denote a string that needs to be sanitized and by S a set of *sensitive patterns* (i.e., strings modeling confidential information). Any substring of W that is not in S may be referred to as *nonsensitive pattern*.

By $\mathcal{R}$ we denote a reference string drawn from the same probability distribution of substring occurrences as W. That is, each sensitive pattern and each string used to replace a sensitive pattern occur with the same probability in $\mathcal{R}$ as in W. $\mathcal{R}$ is known to both the user and the attacker, while W is only known to the user. Therefore, $\mathcal{R}$ can be used by the attacker to derive the probability that any substring u (including a sensitive pattern) occurs in a certain "part" of W, which we define as l-context (or simply *context*) below.

Definition 1 (l-context)**.** *Given a nonnegative integer l and a reference string $\mathcal{R}$, an l-context c_l of a substring u in $\mathcal{R}$ is a pair of strings (c_l^ℓ, c_l^r) such that the string $c_l^\ell \cdot u \cdot c_l^r$ occurs in $\mathcal{R}$ and $|c_l^\ell| = |c_l^r| = l$.*

Note that a substring u may have multiple contexts in $\mathcal{R}$. Also, note that Definition 1 has a parameter l that controls the length of the substrings c_l^ℓ and c_l^r of $\mathcal{R}$ occurring right before and right after u. This offers the flexibility to model attackers with different powers (a larger l implies a more powerful attacker) and is in line with research on mobility mining with Markov chains of order l [50]. The following example illustrates the notion of context.

Example 3. *When the reference string $\mathcal{R}$ is* GGACTTACGGAATTCCGGCATTCAGG *and $l = 2$, the substring $u =$* AC *of $\mathcal{R}$ has two 2-contexts:* (GG, TT) *and* (TT, GG).

The contexts in our approach are given as input by the user. The search for contexts in a reference string $\mathcal{R}$ can be performed efficiently with well-known pattern matching algorithms [51]. The main idea is to find each occurrence of a sensitive pattern in $\mathcal{R}$ and then assign to c_l^ℓ (respectively, c_l^r) the l letters that are right before (respectively, right after) this occurrence of u; see Section II in Supplemental Material for details. We denote by C the set of all such contexts.

B. Constructing the Input Data to the Algorithms

Replaceable and Replacement Strings. Our mathematical programming algorithms are applied to a single context c_l, and they create a replacement matrix P whose rows correspond to the set U_{c_l} of strings that may be replaced and columns to the set U'_{c_l} of strings that can be used as replacements. Then, they compute a replacement probability $P(u \mid u')$ for each row-column pair (u, u') in P. In the following, we explain how to construct the sets U_{c_l} and U'_{c_l}.

We first add all sensitive patterns that are associated with c_l (i.e., each string $u \in S$ such that $c_l^\ell \cdot u \cdot c_l^r$ occurs in $\mathcal{R}$) into U_{c_l} and into U'_{c_l}. This is to allow any such pattern to be replaced by any other sensitive pattern that is associated with c_l or by itself (i.e., not replaced). We want to ensure that (I) there are sufficiently many potential replacements for each sensitive pattern, as this improves privacy, and (II) there are not "too" many potential replacements, as this would increase the runtime of our mathematical programming algorithms. Thus, we require that any sensitive pattern $u \in U_{c_l}$ can be replaced by any of K nonsensitive patterns, where K is a parameter specified by the user.

To select the K nonsensitive patterns, we observe that the quality loss of replacing a pattern $u \in U_{c_l}$ with a pattern $u' \in U'_{c_l}$ can be quantified by the *distance* $d(u, u')$, where $d()$ is any string distance function; our mathematical programming algorithms can work with any $d()$. For example, one can use measures such as Hamming distance or edit distance [51], as well as semantic-aware distances [52]–[54]. Thus, we select every nonsensitive pattern u' whose distance $\sum_{u \in U_{c_l}} d(u', u)$ from the (sensitive) patterns in U_{c_l} is one of the K smallest (breaking ties arbitrarily). These K nonsensitive patterns are added to U_{c_l} and U'_{c_l} so that each can replace any other pattern U_{c_l} or itself (i.e., not be replaced). Thus, $U_{c_l} = U'_{c_l}$. However, for clarity, we will use U_{c_l} to refer to the strings that may be replaced and U'_{c_l} to those that may replace them.

Based on the above, we define the set U'_{c_l} as follows:

Definition 2 (Set of Replacement Strings)**.** *The set U'_{c_l} is the union of the set $\{u \mid u \in S \wedge |\mathtt{occ}_{\mathcal{R}}(c_l^\ell \cdot u \cdot c_l^r)| > 0\}$ of sensitive patterns that are associated to c_l and the set of K nonsensitive patterns in $\mathcal{R}$ such that each such pattern u has one of the K smallest $\sum_{u \in U_{c_l}} d(u', u)$.*

The main idea for constructing U'_{c_l} is (I) to find the occurrences of c_l^ℓ and of c_l^r in $\mathcal{R}$, using a pattern matching algorithm [51]; (II) for each occurrence, to find as a candidate u' the string right after c_l^ℓ and right before c_l^r, and compute its distance $\sum_{u \in U_{c_l}} d(u', u)$; and (III) after candidate creation, to keep the best K candidates. See Section III of the Supplemental Material for details.

Example 4. *Consider the reference string $\mathcal{R} =$* GGACTTACGGAATTCCGGCATTCAGG *from Example 3 and that $S =$* {AC} *and $K = 1$.* AC *has two 2-contexts* (GG, TT) *and* (TT, GG)*. To construct $U'_{(\mathtt{GG},\mathtt{TT})}$, we first add the sensitive pattern* AC *to this set and to $U_{(\mathtt{GG},\mathtt{TT})}$. There are two nonsensitive candidates for being added into $U'_{(\mathtt{GG},\mathtt{TT})}$;* AA *and* CA *(underlined in $\mathcal{R}$). We use the Hamming distance as $d()$, and compute $\sum_{u \in U_{(\mathtt{GG},\mathtt{TT})}} d(\mathtt{AA}, u) = d(\mathtt{AA}, \mathtt{AC}) = 1$ and $\sum_{u \in U_{(\mathtt{GG},\mathtt{TT})}} d(\mathtt{CA}, u) = d(\mathtt{CA}, \mathtt{AC}) = 2$. Thus, we add* AA *into $U'_{(\mathtt{GG},\mathtt{TT})}$ and hence $U'_{(\mathtt{GG},\mathtt{TT})} = \{\mathtt{AC}, \mathtt{AA}\}$. Similarly, $U'_{(\mathtt{TT},\mathtt{GG})}$ is $\{\mathtt{AC}, \mathtt{CC}\}$.*

Prior Probabilities. Let $\pi(u \mid c_l)$ be the probability that string $u \in U_{c_l}$ occurs in the reference string $\mathcal{R}$ conditional on c_l. Our algorithms take as input $\pi(u \mid c_l)$, for each $u \in U_{c_l}$. This probability is part of the attacker's background knowledge and is defined below.

Definition 3 (Prior probability of u conditional on the l-context)**.** *The prior probability of string $u \in U_{c_l}$ conditional on the l-context $c_l = (c_l^\ell, c_l^r)$ is $\pi(u \mid c_l) =$*

$\frac{|\mathsf{occ}_{\mathcal{R}}(c_l^\ell \cdot u \cdot c_l^r)|}{\sum_{v \in U_{c_l}} |\mathsf{occ}_{\mathcal{R}}(c_l^\ell \cdot v \cdot c_l^r)|}$.

To compute this probability, we search for each occurrence of $c_l^\ell \cdot u \cdot c_l^r$ in $\mathcal{R}$ using a pattern matching algorithm based on ideas from Section 4.2 of [51]; see Algorithm 1. Our algorithm uses two classic text indexes, the suffix array (SA) and the longest common prefix (LCP) array [55]. It first constructs the SA and LCP array of $\mathcal{R}$, and then finds the starting positions of the occurrences of $c_l^\ell \cdot u \cdot c_l^r$ in $\mathcal{R}$, by matching $c_l^\ell \cdot u \cdot c_l^r$ using the SA and LCP array. Next, it stores the size $|\mathsf{occ}_{\mathcal{R}}(c_l^\ell \cdot u \cdot c_l^r)|$ of these positions and adds it to a variable den, which becomes equal to the denominator of $\pi(u \mid c_l)$ after the for-loop terminates. The final output $\pi(u \mid c_l)$ is computed for each u by dividing the stored size $|\mathsf{occ}_{\mathcal{R}}(c_l^\ell \cdot u \cdot c_l^r)|$ by den.

The time complexity of Algorithm 1 is $O(|\mathcal{R}|+|U_{c_l}|\cdot(|c_l^\ell \cdot u \cdot c_l^r| + \log|\mathcal{R}|))$, where u is the string with the maximum length in U_{c_l}, as we need $O(|\mathcal{R}|)$ time to construct the SA and LCP of $\mathcal{R}$ and then execute the pattern matching algorithm in [51] $|U_{c_l}|$ times.

Algorithm 1: Computing $\pi(u \mid c_l)$ for each u in U_{c_l}

Input : $\mathcal{R}$, U_{c_l}, $c_l = (c_l^\ell, c_l^r)$
Output: $\pi(u \mid c_l)$, for each $u \in U_{c_l}$

Construct the SA and LCP array of $\mathcal{R}$
$den \leftarrow 0$
for $u \in U_{c_l}$ **do**
  $|\mathsf{occ}_{\mathcal{R}}(c_l^\ell \cdot u \cdot c_l^r)| \leftarrow$ Use the SA and LCP array of $\mathcal{R}$ to count the number of occurrences of $c_l^\ell \cdot u \cdot c_l^r$ in $\mathcal{R}$
  $den \leftarrow den + |\mathsf{occ}_{\mathcal{R}}(c_l^\ell \cdot u \cdot c_l^r)|$
output $\pi(u \mid c_l) \leftarrow \frac{|\mathsf{occ}_{\mathcal{R}}(c_l^\ell \cdot u \cdot c_l^r)|}{den}$, for each $u \in U_{c_l}$

Quality Loss and Privacy Gain Matrix. Our algorithms take as input the matrices d_q and d_p, which quantify, respectively, the quality loss and privacy gain from string replacements.

The matrix d_q is a $|U'_{c_l}| \times |U_{c_l}|$ nonnegative matrix whose elements $d_q(u', u)$ determine the quality loss incurred when replacing a string $u \in U_{c_l}$ by a string $u' \in U'_{c_l}$, where u and u' are not necessarily distinct. The value of element $d_q(u', u)$ is equal to the same distance $d(u', u)$ between u' and u that we used to select the K nonsensitive patterns earlier in this section.

Let $\widehat{U}_{c_l}$ be the set of strings that an attacker guesses when observing the replacements of the strings in U_{c_l}. In practice, we set $\widehat{U}_{c_l} = U_{c_l}$. The matrix d_p is a $|\widehat{U}_{c_l}| \times |U_{c_l}|$ nonnegative matrix whose element $d_p(\hat{u}, u)$ quantifies the privacy gain when the attacker's guess is $\hat{u}$ and the actual string is u.

The values of the d_p matrix are provided by the user. However, we provide some intuition for how these values may be chosen. There are 6 cases for a pair $(\hat{u}, u)$ in d_p:

I $u \neq \hat{u}$, $u \in S, \hat{u} \in S$.
II $u \neq \hat{u}$, $u \in S$, $\hat{u} \notin S$.
III $u \neq \hat{u}$, $u \notin S$, $\hat{u} \in S$.
IV $u \neq \hat{u}$, $u \notin S$, $\hat{u} \notin S$.
V $u = \hat{u}$, $u \in S$, $\hat{u} \in S$.
VI $u = \hat{u}$, $u \notin S$, $\hat{u} \notin S$.

In Cases V and VI, the attacker *correctly* guesses u, so $d_p(\hat{u}, u)$ will take its minimum value 0. In Cases I and II, the attacker does *not* guess the *sensitive* pattern u correctly, so $d_p(\hat{u}, u)$ may take a positive value, since u is sensitive; and in Case I a larger value than Case II, since in Case I u is guessed as sensitive. In Cases III and IV, the attacker does *not* guess the *nonsensitive* pattern u correctly, so $d_p(\hat{u}, u)$ may take a small positive value (smaller than in Cases I and II), since u does not represent confidential information.

C. Attacker Model

The attacker's objective is to minimize privacy (i.e., the opposite of the privacy goal of each corresponding sanitization algorithm in Section III-D). We assume that the attacker knows the reference string $\mathcal{R}$. Also, for each context c_l, the attacker knows the P matrix that is output by any of our sanitization algorithms, the prior probability $\pi(u \mid c_l)$ for each $u \in U_{c_l}$, the privacy gain matrix d_p, the quality loss matrix d_q, and the set $\widehat{U}_{c_l}$.

D. Mathematical Programming Sanitization Algorithms

In different application scenarios, the goal of sanitization may be different [11], [21]. One may want to gain as much privacy as possible, as long as the quality loss is acceptable, or they may prefer to have a minimum quality loss, as long as the privacy gain is acceptable. To handle these scenarios, we design three optimal, mathematical programming algorithms with different objectives and constraints. For an introduction to mathematical programming, we refer the reader to [56]. All algorithms are applied to a single context c_l and output a matrix P of optimal replacement probabilities. To sanitize a string, we need to apply them to each context *separately*.

APG-LP. Our first algorithm maximizes the Average Privacy Gain (APG) and it is based on Linear Programming (LP). APG-LP limits the average quality loss caused by string replacement. We show how to formulate this algorithm as a linear program (see Program 1). The other algorithms we present later can be formulated analogously.

We focus on a single context c_l to which the algorithm is applied. Consider the case when the attacker observes the string $u' \in U'_{c_l}$ together with c_l somewhere in the sanitized string. We will show how to compute APG step by step. First, we compute the probability that the string that has been replaced is u when the attacker observes a replacement string u' in the sanitized string (Eq. 1). To guess the string $u \in U_{c_l}$, the attacker can form the posterior distribution on u, conditional on the observed replacement u':

$$\begin{aligned} \Pr(u \mid u') &= \frac{\Pr(u, u')}{\Pr(u')} \\ &= \frac{\Pr(u' \mid u)\Pr(u)}{\sum_u \Pr(u, u')} \\ &= \frac{P(u' \mid u)\pi(u \mid c_l)}{\sum_u P(u' \mid u)\pi(u \mid c_l)}, \end{aligned} \tag{1}$$

where we have used our definitions that $\Pr(u' \mid u) = P(u' \mid u)$ and $\Pr(u) = \pi(u \mid c_l)$.

When the attacker observes a string u' in the sanitized string, they estimate the string that has been replaced as $\hat{u} \in \widehat{U}_{c_l}$. The privacy gain for a specific estimate $\hat{u}$ of u is $d_p(\hat{u}, u)$, and the probability that this estimate is made is $\Pr(u \mid u')$. Averaging over all $u \in U_{c_l}$, we obtain the user's *conditional expected privacy* for an arbitrary $\hat{u} \in \widehat{U}_{c_l}$:

$$\sum_u \Pr(u \mid u') d_p(\hat{u}, u).$$

The attacker's objective is then to choose an estimate string $\hat{u} \in \widehat{U}_{c_l}$ to minimize the user's conditional expected privacy, where the expectation is taken over $\Pr(u \mid u')$. This leads to the following equation:

$$\min_{\hat{u}} \sum_u \Pr(u \mid u') d_p(\hat{u}, u). \tag{2}$$

Since the privacy for a string $u' \in U'_{c_l}$ is given by Eq. 2 and this u' is selected as a replacement with probability $\Pr(u') = \sum_u P(u' \mid u)\pi(u \mid c_l)$, averaging over all possible $u' \in U'_{c_l}$ leads to the user's *unconditional expected privacy*:

$$\begin{aligned} &\sum_{u'} \Pr(u') \min_{\hat{u}} \sum_u \Pr(u \mid u') d_p(\hat{u}, u) \\ &= \sum_{u'} \min_{\hat{u}} \sum_u \pi(u \mid c_l) P(u' \mid u) d_p(\hat{u}, u), \end{aligned} \tag{3}$$

where we have used $\Pr(u')\Pr(u \mid u') = \Pr(u, u') = \Pr(u)\Pr(u' \mid u) = \pi(u \mid c_l)P(u' \mid u)$.

We define the *Minimum Privacy Gain* (MPG) as follows:

$$x_{u'} \triangleq \min_{\hat{u}} \sum_u \pi(u \mid c_l) P(u' \mid u) d_p(\hat{u}, u). \tag{4}$$

Incorporating $x_{u'}$ into Eq. 3, we rewrite the unconditional expected privacy of the user as

$$\sum_{u'} x_{u'}, \tag{5}$$

which the user aims to maximize by choosing the optimal P matrix. We refer to Eq. 5 as *Average Privacy Gain* (APG). The minimum operator would make our mathematical program nonlinear. As this is undesirable for efficiency reasons, we transform Eq. 4 into a series of linear constraints, so that the program becomes linear, following [57]:

$$x_{u'} \leq \sum_u \pi(u \mid c_l) P(u' \mid u) d_p(\hat{u}, u), \ \forall \hat{u}. \tag{6}$$

Then, maximizing Eq. 5 under Eq. 4 is equivalent to maximizing Eq. 5 under Eq. 6. Therefore, we formulate the privacy objective for the user from Eq. 5 (as a maximization objective) and from Eq. 6 (one constraint for each $u' \in U'_{c_l}$).

We now formulate the user's quality goal as a constraint on the average quality loss caused by the replacement of $u \in U_{c_l}$ by $u' \in U'_{c_l}$. The probability that u appears in the string is $\Pr(u) = \pi(u \mid c_l)$, and the probability that u' replaces u is $\Pr(u' \mid u) = P(u' \mid u)$. The quality loss caused by replacing u by u' is $d_q(u', u)$. Therefore, the *Average Quality Loss* (AQL) over all possible pairs $(u, u') \in U_{c_l} \times U'_{c_l}$ is

$$\sum_u \pi(u \mid c_l) \sum_{u'} P(u' \mid u) d_q(u', u). \tag{7}$$

We want AQL to be at most equal to a user-specified threshold Q_{MAX} in our mathematical program.

Thus, APG-LP can be formulated as in Program 1. Eq. 9 are the series of linear constraints from Eq. 6, one series for each value of u'; Eq. 10 reflects the quality loss constraint; and Eq. 11 and Eq. 12 reflect that, for each value of u, its corresponding row $P(u' \mid u)$, $\forall u' \in U'_{c_l}$, is a probability distribution. Then, the algorithm finds a solution $P(u' \mid u), x_{u'}, \forall u \in U_{c_l}, u' \in U'_{c_l}$ to Program 1.

$$\textbf{Maximize} \quad \sum_{u'} x_{u'} \tag{8}$$

subject to

$$x_{u'} \leq \sum_u \pi(u \mid c_l) P(u' \mid u) d_p(\hat{u}, u), \ \forall \hat{u}, u' \tag{9}$$

$$\sum_u \pi(u \mid c_l) \sum_{u'} P(u' \mid u) d_q(u', u) \leq Q_{\text{MAX}} \tag{10}$$

$$\sum_{u'} P(u' \mid u) = 1, \ \forall u \tag{11}$$

$$P(u' \mid u) \geq 0, \ \forall u, u' \tag{12}$$

Program 1: APG-LP: Average-privacy-gain-maximizing, average-quality-constrained linear program.

AQL-MILP. This algorithm is derived from a nonlinear program (Program 2), which we linearize to improve its runtime without changing its output. The algorithm differs from APG-LP in that it minimizes the Average Quality Loss (AQL) in Eq. 13 instead of upper-bounding it, while lower-bounding the privacy gain *for each* possible replacement $u' \in U'_{c_l}$ by a user-specified threshold T_{MIN} (Eq. 14), instead of maximizing APG. We write *possible* replacement because if a string u' has $\sum_u P(u' \mid u) = 0$ (i.e., it is not used to replace any $u \in U_{c_l}$) then its privacy gain does not need to be constrained. This is achieved by Eq. 14, where we use a constant $M > \frac{T_{\text{MIN}}}{\min_{u' \in U'_{c_l} : \sum_u P(u'|u) > 0} \sum_u P(u'|u)}$. This equation ensures that, for any u' with $\sum_u P(u' \mid u) > 0$, its privacy gain must be at least T_{MIN}, while for any other u', its privacy gain must only be at least 0. Eqs. 15 and 16 are the same as the last two equations in APG-LP.

Program 2 is clearly nonlinear due to the minimum operator in Eq. 14. We therefore linearize it as shown in Program 3. For this, we add a continuous decision variable $z_{u'}$, binary decision variables $y_{u',1}$, $y_{u',2}$, and constraints in Eqs. 19 to 24. Moreover, we also introduce a constant m satisfying $m > T_{\text{MIN}}$ and $m > M \cdot \sum_u P(u' \mid u)$, for any $u' \in U'_{c_l}$. The proof that the linearization is correct is in Section IV of the Supplemental Material. Note that the existence of integer and continuous variables make Program 3 a mixed-integer linear program (MILP).

DP-LP. Differential privacy (DP) [26] is a rigorous privacy principle to prevent an attacker from distinguishing between

$$\textbf{Minimize} \quad \sum_{u} \pi(u \mid c_l) \sum_{u'} P(u' \mid u) d_q(u', u) \tag{13}$$

subject to

$$\sum_{u} \pi(u \mid c_l) P(u' \mid u) d_p(\hat{u}, u) \geq \min\{T_{\text{MIN}}, M \cdot \sum_{u} P(u' \mid u)\}, \forall \hat{u}, u' \tag{14}$$

$$\sum_{u'} P(u' \mid u) = 1, \forall u \tag{15}$$

$$P(u' \mid u) \geq 0, \forall u, u' \tag{16}$$

Program 2: Average-quality-loss-minimizing, privacy-constrained nonlinear program.

$$\textbf{Minimize} \quad \sum_{u} \pi(u \mid c_l) \sum_{u'} P(u' \mid u) d_q(u', u) \tag{17}$$

subject to

$$z_{u'} \leq \sum_{u} \pi(u \mid c_l) P(u' \mid u) d_p(\hat{u}, u), \forall \hat{u}, u' \tag{18}$$

$$z_{u'} \leq T_{\text{MIN}}, \forall u' \tag{19}$$

$$z_{u'} \geq T_{\text{MIN}} - m \cdot y_{u',1}, \forall u' \tag{20}$$

$$z_{u'} \leq M \cdot \sum_{u} P(u' \mid u), \forall u' \tag{21}$$

$$z_{u'} \geq M \cdot \sum_{u} P(u' \mid u) - m \cdot y_{u',2}, \forall u' \tag{22}$$

$$y_{u',1} + y_{u',2} \leq 1, \forall u' \tag{23}$$

$$y_{u',1}, y_{u',2} \in \{0, 1\}, \forall u' \tag{24}$$

$$\sum_{u'} P(u' \mid u) = 1, \forall u \tag{25}$$

$$P(u' \mid u) \geq 0, \forall u, u' \tag{26}$$

Program 3: AQL-MILP: Average-quality-loss-minimizing, privacy-constrained linear program.

different inputs. In our work, the inputs are any two strings from U_{c_l}, and DP can be defined as follows:

Definition 4. *Let $\mathcal{K}$ be the replacement operator that takes as input a string from U_{c_l} and outputs a set of possible replacements from U'_{c_l} together with the probabilities of these replacements. The $\mathcal{K}$ operator offers ϵ-differential privacy, for a positive real number ϵ, if for all strings u_i, $u_j \in U_{c_l}$ and all subsets L of U'_{c_l}, it holds that:*

$$\Pr[\mathcal{K}(u_i) \in L] \leq \mathrm{e}^{\epsilon} \cdot \Pr[\mathcal{K}(u_j) \in L]. \tag{27}$$

Based on Definition 4, we propose our DP-LP linear programming algorithm (see Program 4). In this algorithm, $\mathcal{K}$ corresponds to the P matrix, i.e., the choice of replacements $u' \in U'_{c_l}$ for every $u \in U_{c_l}$, while L corresponds to a single replacement u'. Therefore, Eq. 27 is directly written as the constraints in Eq. 28 using the notation in our algorithm; for a given context c_l, the inequality

$$P(u' \mid u_i) \leq P(u' \mid u_j) \cdot \mathrm{e}^{\epsilon} \tag{28}$$

must hold for all output strings $u' \in U'_{c_l}$, and for all input pairs of (u_i, u_j), $u_i, u_j \in U_{c_l}$ with a given positive real number ϵ. The objective function in Eq. 29 and the constraints in Eq. 31 and Eq. 32 are as in AQL-MILP.

$$\textbf{Minimize} \quad \sum_{u} \pi(u \mid c_l) \sum_{u'} P(u' \mid u) d_q(u', u) \tag{29}$$

subject to

$$P(u' \mid u_i) \leq P(u' \mid u_j) \cdot \mathrm{e}^{\epsilon}, \forall u', u_i, u_j \tag{30}$$

$$\sum_{u'} P(u' \mid u) = 1, \forall u \tag{31}$$

$$P(u' \mid u) \geq 0, \forall u, u' \tag{32}$$

Program 4: DP-LP: Differentially private average-quality-loss-minimizing linear program.

E. Constructing the Sanitized String Z

We first initialize Z with W. Then, for each context c_l, we get the output P matrix from any of our mathematical programming algorithms, and sanitize each string $u \in U_{c_l}$ that appears in Z by selecting a replacement string u' for u with probability $P(u' \mid u)$ (i.e., we sample u' from the probability distribution in the row of matrix P for u).

IV. Experiments

A. Datasets

We use five publicly available datasets commonly used in the string privacy literature [10]–[12], [58]: Trucks (*trucks*) [29], the complete genome of Escherichia coli (*ecoli*) [27], MSNBC (*msnbc*) [28], Kasandr (*kasandr*) [30], and IoT (*iot*) [31]. The *trucks* dataset contains transportation data, *ecoli* contains genomic data, *msnbc* contains clickstream data, *kasandr* contains eCommerce behavior data, and *iot* contains advertisement / traces generated from Bluetooth Low Energy beacons. The dataset characteristics and the default values for the parameters are shown in Table II.

Dataset	Length n	Alphabet size $\lvert\Sigma\rvert$	Size of set S	Sens. pattern length k	Context length l	No. of nonsens. replacements K
trucks	5,763	100	300	5	1	5
ecoli	4,641,652	4	300	15	3	6
msnbc	4,698,764	17	300	10	3	6
kasandr	16,118,213	95	300	10	4	10
iot	18,673,095	63	300	10	4	10

TABLE II: Dataset characteristics and the default values used in our experiments.

B. Experimental Setup

Reference String $\mathcal{R}$. We chose as the reference string $\mathcal{R}$ the input string W which will be sanitized. This is the worst case where the attacker's prior information is completely accurate. The same assumption was made in [21]. That is, both the user and the attacker know the prior probabilities $\pi(u \mid c_l)$, for each context c_l in W.

If the attacker has partial information about $\mathcal{R}$, then the attacker's inference of the sensitive patterns will have errors, so the user's privacy gain will be higher than what we report.

If $\mathcal{R}$ is the same for both the user and the attacker, but it has different statistics from W, then the computation of the privacy gain and of the quality loss as computed by our framework (which is based on the statistics of $\mathcal{R}$) can be inaccurate in either a positive or a negative way. That is, in reality, the privacy gain and quality loss can be better or worse than what our framework reports. The correct values could then be computed by plugging the true statistics in Eq. (7) for quality loss or Eqs. (4) and (5) for privacy gain.

Privacy Gain Matrix d_p. Recall that the values of the privacy gain matrix d_p must be specified by the user. In our experiments, we set these values as follows – see Section III-B for the six cases (types of privacy gain): Case I, $d_p(\hat{u}, u) = 9$; Case II, $d_p(\hat{u}, u) = 10$; Case III, $d_p(\hat{u}, u) = 2$; Case IV, $d_p(\hat{u}, u) = 1$; and Cases V and VI, $d_p(\hat{u}, u) = 0$.

Privacy Gain and Quality Loss Metrics. To evaluate the effectiveness of a sanitization algorithm, we measure the privacy gained against the strategic attacker, as well as the quality loss caused by the algorithm:

1. *Average Privacy Gain (APG) for Context c_l.* This is the measure from Eq. 5 which we denote by $P_{\text{avg}}(c_l)$, for context c_l.

2. *Weighted Average Privacy Gain (WAPG) for Sensitive Set S.* This measure aggregates the APG scores of all contexts in the context set C, weighting each context $c_l \in C$ by the total number $f(c_l)$ of occurrences of the sensitive patterns associated to c_l in W. The intuition is that a context with a larger $f(c_l)$ should matter more when we measure the privacy gain. Thus, WAPG is defined as

$$\sum_{c_l \in C} \frac{f(c_l) \cdot P_{\text{avg}}(c_l)}{\sum_{c_l \in C} f(c_l)}.$$

3. *Weighted Minimum Privacy Gain (WMPG) for Sensitive Set S.* The minimum privacy gain (MPG) for context c_l is defined as $P_{\text{MIN}}(c_l) = \min_{u': \sum_u P(u'|u) > 0} x_{u'}$, where $x_{u'}$ is as in Eq. 4. The intuition is to calculate the privacy gain based on the protection offered when $u \in U_{c_l}$ is replaced by the string $u' \in U'_{c_l}$ that offers the smallest privacy gain. Based on that, the WMPG measure is defined as

$$\sum_{c_l \in C} \frac{f(c_l) \cdot P_{\text{MIN}}(c_l)}{\sum_{c_l \in C} f(c_l)},$$

where $f(c_l)$ is as defined in WAPG.

4. *Average Quality Loss (AQL) for Context c_l.* This is the measure from Eq. 7, which we denote by $Q_{\text{avg}}(c_l)$, for context c_l.

5. *Weighted Average Quality Loss (WAQL) for Sensitive Set S.* This measure aggregates the AQL scores of all contexts in the context set C, weighting each context $c_l \in C$ by a weight $w(c_l) = \frac{\sum_{u \in U_{c_l}} |\mathsf{occ}_W(u)|}{\sum_{c_l \in C} \sum_{u \in U_{c_l}} |\mathsf{occ}_W(u)|}$. The intuition is that a context with a larger $w(c_l)$ should matter more when we measure the quality loss. Thus, WAQL is defined as

$$\sum_{c_l \in C} \frac{w(c_l) \cdot Q_{\text{avg}}(c_l)}{\sum_{c_l \in C} w(c_l)}.$$

6. *q-Gram Distance (D_q).* We quantify the quality loss, caused by sanitizing the string W, using the well-known q-gram distance [59]. Given an integer q, we denote by Σ^q the set of all possible length-q strings (q-grams) over Σ, and fix an enumeration order to list the elements in Σ^q. Then, the *q-gram profile* of string W is the vector $G_q(W) = (|\mathsf{occ}_W(x)|, x \in \Sigma^q)$. The q-gram distance between the original string W and the corresponding sanitized string Z is defined as

$$D_q(W, Z) = \sum_{x \in \Sigma^q} \Big| G_q(W)[x] - G_q(Z)[x] \Big|. \tag{33}$$

7. *JS-divergence (JS).* We quantify the loss in predictive performance of the original string W when sanitized using the well-known JS-divergence measure [60]. Given two probability distributions p_1, p_2, the JS-divergence between them is defined as

$$\text{JS}(p_1, p_2) = \frac{1}{2} \cdot \sum_{i \in [1,n]} \left(p_1[i] \cdot \log_2 \left(\frac{2 \cdot p_1[i]}{p_1[i] + p_2[i]} \right) + p_2[i] \cdot \log_2 \left(\frac{2 \cdot p_2[i]}{p_1[i] + p_2[i]} \right) \right). \tag{34}$$

Specifically, we quantify the predictive performance based on (first-order) Markov-chain prediction: First, we construct the transition matrix $\mathcal{M}$ from the original string W and the corresponding transition matrix $\mathcal{M}'$ from the sanitized string Z. Then we compute the JS-divergence between the corresponding rows of $\mathcal{M}$ and $\mathcal{M}'$ and compute the average or maximum over all rows. JS-divergence values are in $[0, 1]$ and smaller values mean that the sanitized string can support prediction as well as the original string.

Code and Environment. Our code was written in C++ and is available at https://bit.ly/3WXdNK6. We used the Gurobi solver v. 11.0.0 (single-thread configuration) in our linear programming and mixed-integer linear programming algorithms. All experiments were run on an Intel Core i7-1365U CPU @ 1.50 GHz with 32 GB RAM, with Ubuntu 22.04.4 LTS operating system.

C. Methods

We evaluated our algorithmic framework using each of our algorithms (APG-LP, AQL-MILP, DP-LP). We use the name of the algorithm to refer to the corresponding approach. Since existing string sanitization methods are not comparable to our approach (see Section II), we used as a baseline an algorithm inspired by the basic obfuscation algorithm in [21]. The baseline algorithm, referred to as Baseline, chooses a replacement string $u' \in U'_{c_l}$ for each $u \in U_{c_l}$ uniformly at random, i.e., each u' is chosen with equal probability, among the R closest strings to u that are contained in U'_{c_l} with respect to their distance $d()$,[1] where R

[1]As distance $d()$, we used the Hamming distance due to its efficiency (if for two strings u, u', it holds that $|u| < |u'|$, we append $|u'| - |u|$ occurrences of a letter that is not contained in Σ to u, and similarly we append $|u| - |u'|$ occurrences of this letter to u' if $|u| > |u'|$).

is a parameter set by the user. Thus, the element $P(u' \mid u)$ of the P matrix for the baseline has a value $\frac{1}{R}$ if u' is among the R closest strings to u, and 0 otherwise.

D. Experimental Results

Impact of Main Parameters. As each algorithm has its own parameter to control privacy gain and quality loss (APG-LP has Q_{MAX}, AQL-MILP has T_{MIN}, DP-LP has ϵ, and Baseline has R), we evaluate these algorithms separately. We choose one context randomly to evaluate our algorithms, since they are applied to a single context (similar trends were observed for all other contexts we tested). We present results for *ecoli* in Fig. 2. The results for the other datasets are analogous (see Section V in Supplemental Material).

Recall that the APG-LP algorithm maximizes APG subject to AQL $\leq Q_{\text{MAX}}$ and other constraints. As can be seen in Figs. 2a and 2b, with the increase of the quality threshold Q_{MAX}, both APG and AQL generally increase. This is because higher privacy comes with more quality loss. Also, after some Q_{MAX} value, APG remains the same, as it reaches its maximum possible value subject to the constraints. Increasing Q_{MAX} beyond that value does not lead to a corresponding monotonic increase for AQL: The algorithm can output any of the feasible solutions (those with AQL at most Q_{MAX}) that maximize the privacy gain, and it does not necessarily choose the one with the largest AQL.

Recall that the AQL-MILP algorithm minimizes AQL subject to ensuring that the privacy gain of each possible replacement u' is at least T_{MIN} and other constraints. Figs. 2c and 2d show that both APG and AQL increase to some level when the privacy threshold T_{MIN} increases. This is because a higher T_{MIN} requires the output to have more privacy for each possible replacement - see Eq. 14 - and this generally leads to a higher APG. On the other hand, more quality loss has to be incurred, and this leads to a higher AQL. Notice that unlike APG-LP, which can always find a feasible solution no matter how large Q_{MAX} is, AQL-MILP may not always find a feasible solution when T_{MIN} is too large, as the privacy constraints in Eq. 18 to Eq. 24 may not be satisfiable.

Recall that the DP-LP algorithm minimizes AQL subject to ϵ-differential privacy and other constraints. Figs. 2e and 2f demonstrate that increasing ϵ reduces both APG and AQL. This is expected, since the ratio $\frac{P(u'|u_i)}{P(u'|u_j)}$, for any $u_i, u_j \in U_{c_l}$, can have more possible values when ϵ is larger, and the algorithm selects the values that lead to the minimum AQL. Note that for a very large ϵ, Eq. 30 does not really constrain the P matrix values. This leads to a P matrix with 1s on the diagonal, which means that the algorithm optimizes AQL by always choosing the trivial replacement $u' = u$, which makes both APG and AQL equal to zero.

Recall that the Baseline algorithm selects one of the R possible replacements with uniform probability for each $u \in U_{c_l}$. Figs. 2g and 2h show that both APG and AQL increase when R increases. This is because a higher R leads to more possible replacements for each u, which can achieve more privacy gain but also incur a higher quality loss.

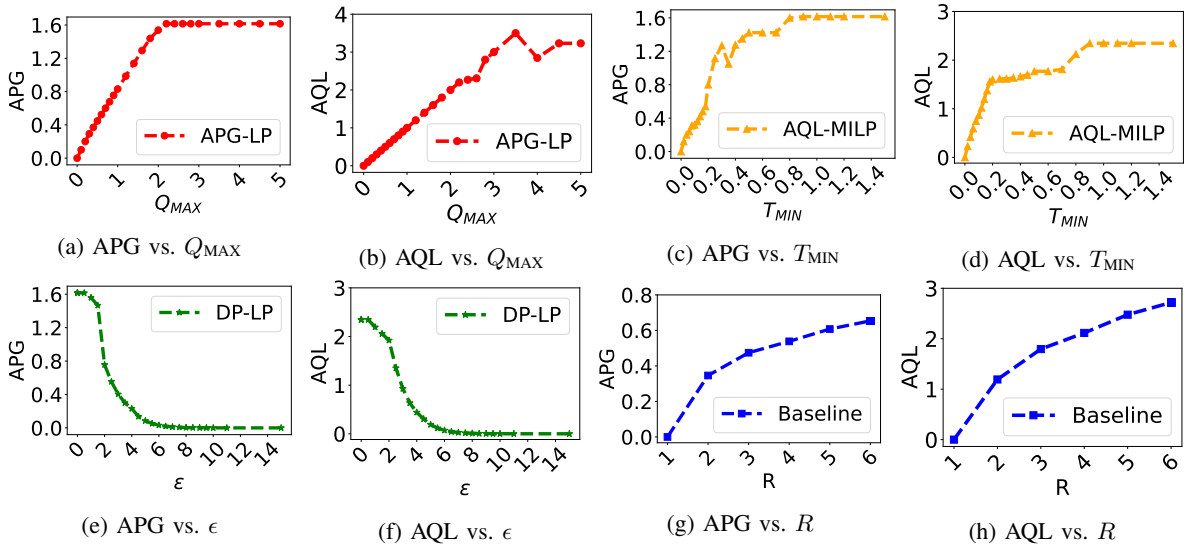
Recall that all algorithms use the parameter K (number of nonsensitive patterns that can replace a sensitive pattern). We evaluated the impact of K on privacy and quality loss. Again, we choose one context randomly (similar trends were observed for all other contexts tested) and present the results for *ecoli* (similar results for other datasets can be found in Section V-A of Supplemental Material).

Fig. 3 shows that a larger K increases privacy (APG increases) for all algorithms and also increases quality loss (AQL increases) for all algorithms except APG-LP and Baseline. As K increases, APG-LP has more potential replacements for the sensitive pattern (hence APG increases); see Fig. 3a. However, as in the experiment of Figs. 2a and 2b, the solver selects the solution among any solution with AQL at most Q_{MAX}, which causes AQL to not only increase or only decrease as K increases. Also, as K increases, U'_{c_l} becomes larger (i.e., there are more potential replacements), and thus AQL-MILP and DP-LP have larger AQL and also larger APG; see Figs. 3c and 3e. For Baseline, we used $R = K$. As K increases, APG increases (see Fig. 3g) since there are more potential replacements, and AQL does not always increase or decrease (see Fig. 3h), as K affects all terms in Eq. 7 in a way that depends on the data and not on how Baseline works. We also evaluated the impact of l (context length); see Section V-A of Supplemental Material.

Privacy Gain and Quality Loss Tradeoff. We examined the effectiveness of our algorithms in terms of the tradeoff between privacy gain and quality loss. To make these algorithms that have different parameters comparable, we configured them to have equal quality loss (or as close to equal as possible) and compared the privacy gain they offer (Fig. 4 and Fig. 5), or vice versa to have the same or as much similar as possible privacy gain and compared the quality loss they incur (Figs. 6, 7, and 8).

We first configured all our algorithms to achieve the same WAQL values, and Baseline to have WAQL values as close as possible to the WAQL values of our algorithms, by trying different R values (it was not possible to have the same WAQL values as the other algorithms, as $R \in \{1, 2, \ldots, |U'_{c_l}|\}$ is the only parameter and we tried all its possible values). In Fig. 4, we show the WAPG of the algorithms for varying WAQL values. APG-LP always outperforms all other algorithms with respect to WAPG. This is because by maximizing APG for each context subject to the constraints, this algorithm also maximizes WAPG (i.e., the weighted average of the privacy gains of all contexts). Moreover, all our algorithms outperform Baseline by achieving a higher privacy gain while incurring a smaller quality loss. For example, our worst-performing algorithm DP-LP achieved on average 76% and up to 109% higher WAPG than Baseline over all the datasets.

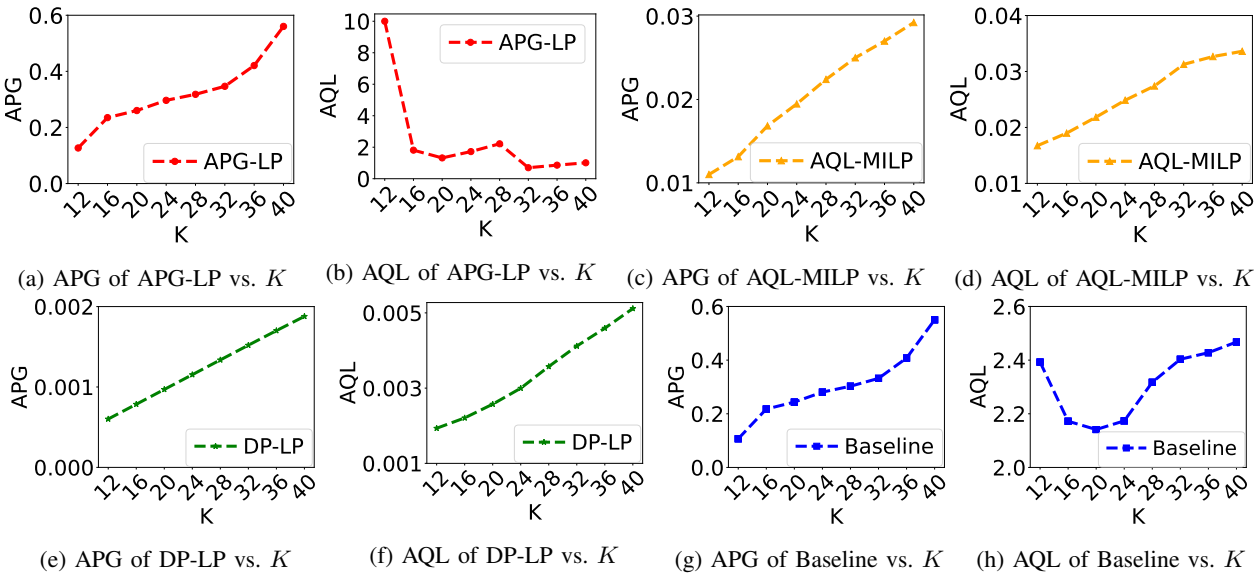
Fig. 5 shows the results of the same experiment as Fig. 4 but for the WMPG measure instead of WAPG. AQL-MILP achieves the highest WMPG among all algorithms. This is because the privacy gain of each possible replacement u' is guaranteed to be at least T_{MIN}, so WMPG (i.e., the weighted



(a) APG vs. Q_{MAX} (b) AQL vs. Q_{MAX} (c) APG vs. T_{MIN} (d) AQL vs. T_{MIN}

(e) APG vs. ϵ (f) AQL vs. ϵ (g) APG vs. R (h) AQL vs. R

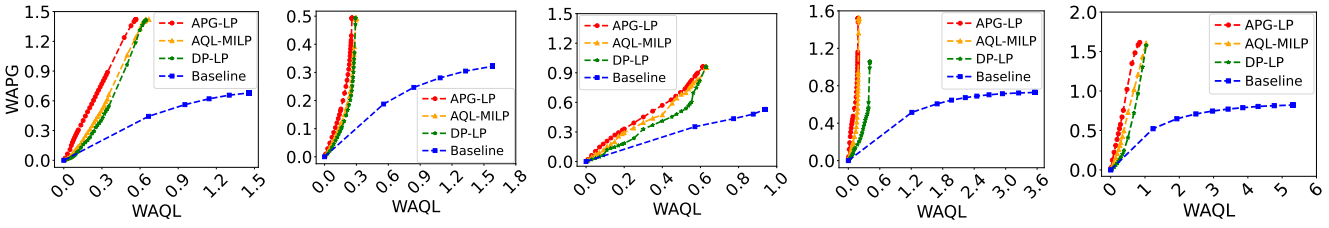
Fig. 2: APG-LP: (a) Average Privacy Gain (APG), (b) Average Quality Loss (AQL) vs. Q_{MAX}. AQL-MILP: (c) APG, (d) AQL vs. T_{MIN}. DP-LP: (e) APG, (f) AQL vs. ϵ. Baseline: (g) APG, (h) AQL vs. R. All results are for the *ecoli* dataset.



(a) APG of APG-LP vs. K (b) AQL of APG-LP vs. K (c) APG of AQL-MILP vs. K (d) AQL of AQL-MILP vs. K

(e) APG of DP-LP vs. K (f) AQL of DP-LP vs. K (g) APG of Baseline vs. K (h) AQL of Baseline vs. K

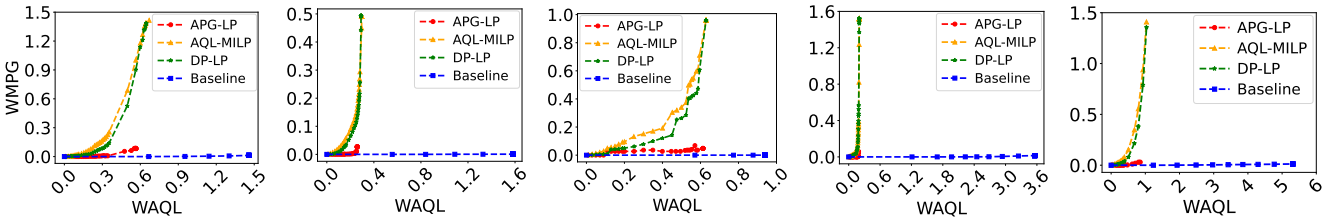
Fig. 3: APG-LP: (a) Average Privacy Gain (APG), (b) Average Quality Loss (AQL) vs. K. AQL-MILP: (c) APG, (d) AQL vs. K. DP-LP: (e) APG, (f) AQL vs. K. Baseline: (g) APG, (h) AQL vs. K. All results are for the *ecoli* dataset.



(a) *msnbc* (b) *ecoli* (c) *trucks* (d) *kasandr* (e) *iot*

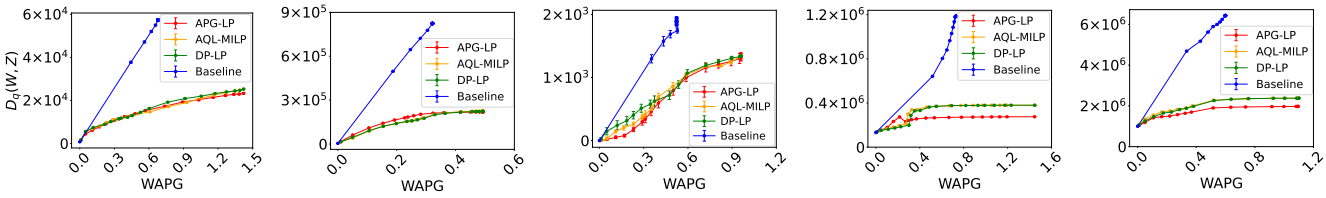
Fig. 4: Weighted Average Privacy Gain (WAPG) for varying Weighted Average Quality Loss (WAQL).

minimum of the privacy gains of all contexts) cannot be very small. The WMPG of DP-LP is close to that of AQL-MILP because DP-LP also focuses on protecting privacy by ensuring that each possible replacement u' satisfies the



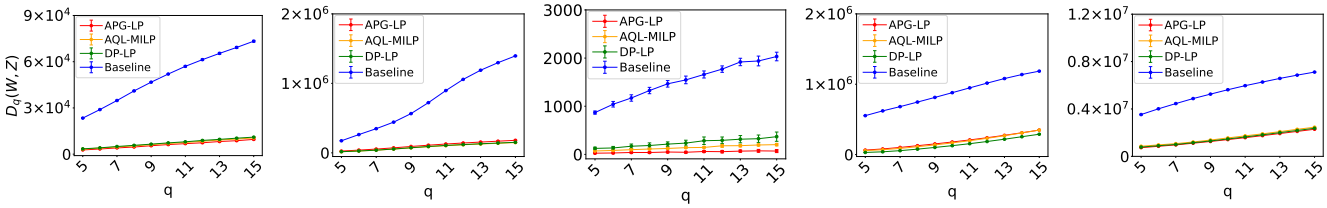
(a) *msnbc* (b) *ecoli* (c) *trucks* (d) *kasandr* (e) *iot*

Fig. 5: Weighted Minimum Privacy Gain (WMPG) for varying Weighted Average Quality Loss (WAQL).



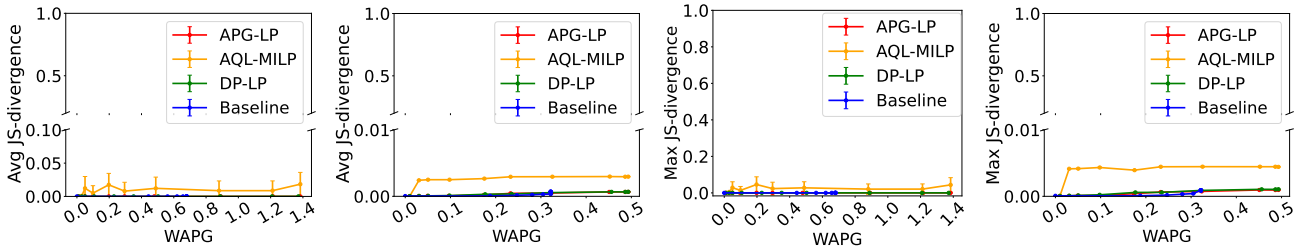
(a) *msnbc* (b) *ecoli* (c) *trucks* (d) *kasandr* (e) *iot*

Fig. 6: q-Gram distance $D_q(W, Z)$ with $q = 10$ for varying Weighted Average Privacy Gain (WAPG). Error bars represent the standard deviation at each point.



(a) *msnbc* (b) *ecoli* (c) *trucks* (d) *kasandr* (e) *iot*

Fig. 7: q-Gram distance $D_q(W, Z)$ for varying q. Error bars represent the standard deviation at each point.



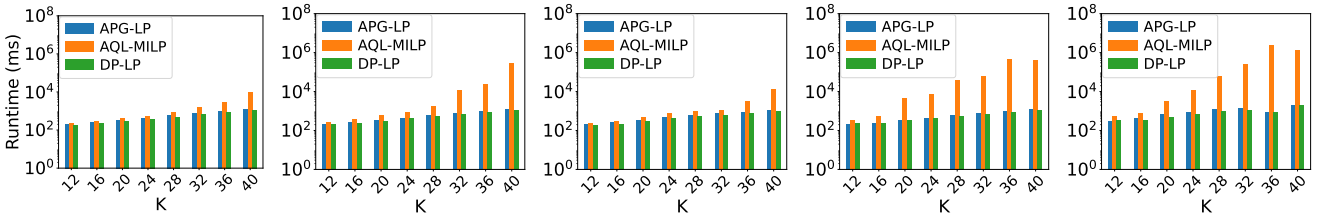
(a) *msnbc* (b) *ecoli* (c) *msnbc* (d) *ecoli*

Fig. 8: Average and Maximum JS-divergence between the corresponding rows of the transition matrix $\mathcal{M}$ on W and the transition matrix $\mathcal{M}'$ on Z for varying WAPG. Error bars represent the standard deviation at each point.

property of DP. APG-LP is worse than the two aforementioned algorithms because it only optimizes the APG over the string replacements, so for some of them the privacy gain can be small. Similarly to Fig. 4, all our algorithms outperform Baseline in both WAQL and WMPG. In fact, the WMPG for Baseline was 0 or very close to 0.

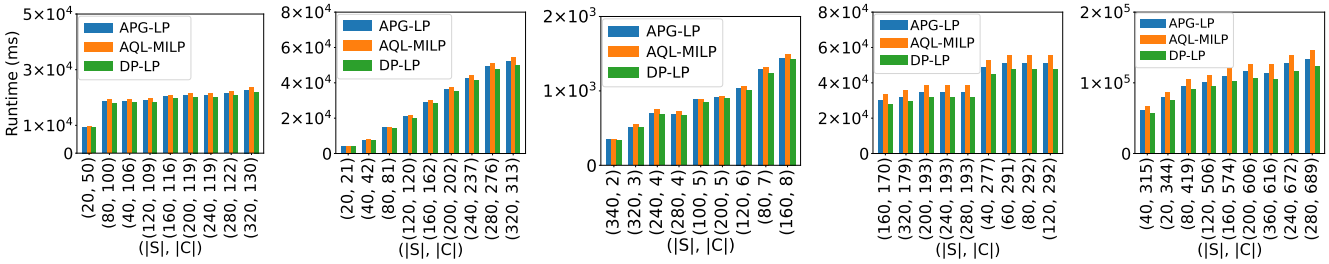
We then configured all our algorithms to achieve the same WAPG values, and Baseline to have as close WAPG values to those of the other algorithms, as before. We use the q-gram distance $D_q(W, Z)$ (see Eq. 33) to measure data quality loss. In Fig. 6, we set $q = 10$, vary WAPG, and compare $D_q(W, Z)$ for all algorithms. In Fig. 7, we set the WAPG of all algorithms to a fixed value 0.11 and vary q to compare their $D_q(W, Z)$. Both Fig. 6 and Fig. 7 show that our algorithms outperform the Baseline, as they achieve lower $D_q(W, Z)$. In particular, our algorithms outperformed Baseline by 216% on average (and by up to 335%).

Last, we evaluate how sanitization affects the predictive performance of the data based on JS-divergence (see Section IV-B). We use the *msnbc* and *ecoli* datasets because



(a) *msnbc* (b) *ecoli* (c) *trucks* (d) *kasandr* (e) *iot*

Fig. 9: Runtime for varying K.



(a) *msnbc* (b) *ecoli* (c) *trucks* (d) *kasandr* (e) *iot*

Fig. 10: Runtime for varying $|S|$.

their alphabet size is smaller and each letter occurs more frequently. Thus, the pairs of letters (transitions) occur more frequently, which means that these datasets are more suitable for making predictions. Figs. 8a and 8b report the average JS-divergence over the pairs of the corresponding rows of the transition matrices $\mathcal{M}$ and $\mathcal{M}'$, while Figs. 8c and 8d report the maximum JS-divergence over the pairs of the corresponding rows of $\mathcal{M}$ and $\mathcal{M}'$. Note that the JS-divergence values are small, which implies that all algorithms preserve the predictive performance of the data fairly well.

Differential Privacy in DP-LP. We show that all other algorithms except DP-LP do *not* satisfy differential privacy for any positive ϵ, unlike DP-LP which satisfies differential privacy for any positive ϵ (as expected). To show this, we rewrote Eq. 30 of DP-LP as follows:

$$\ln\left(\frac{P(u' \mid u_i)}{P(u' \mid u_j)}\right) \leq \epsilon, \forall u', u_i, u_j. \tag{35}$$

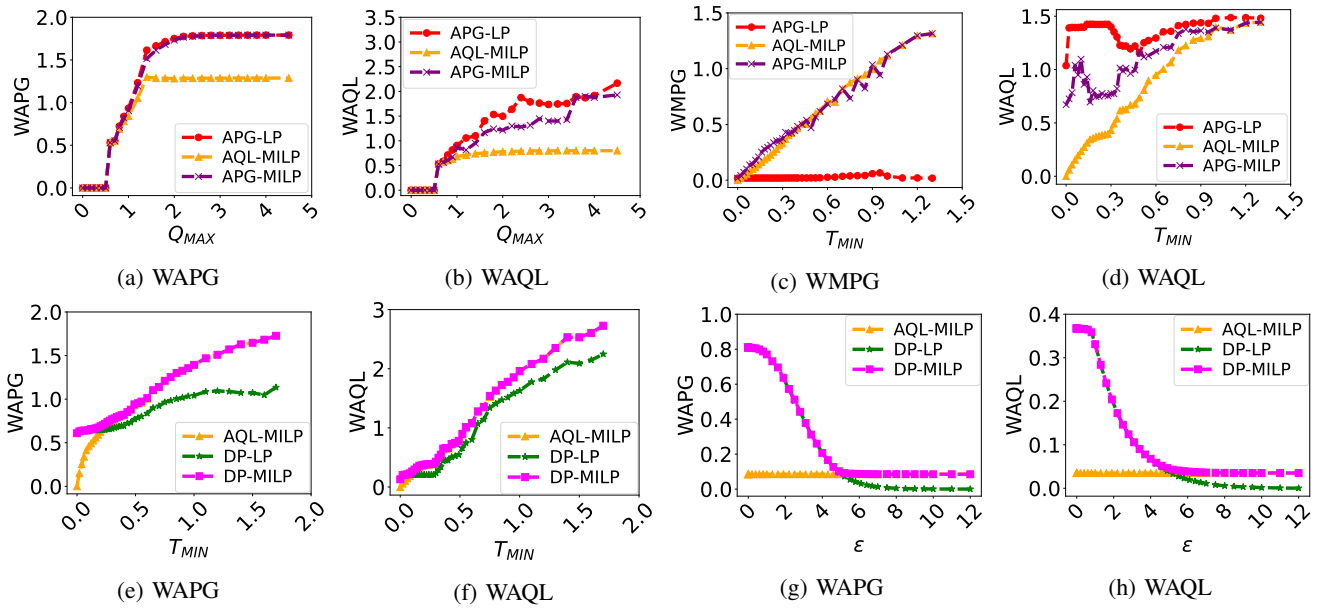
Clearly, if an algorithm is differentially private, then Eq. 35 can always find a feasible solution for any given positive ϵ. We tried large values of ϵ from 2.5 to 10 to examine whether the algorithms other than DP-LP could satisfy differential privacy (although they do not guarantee it), and this did not happen for any of the ϵ tested. Then we computed the maximum ratio $\frac{P(u'|u_i)}{P(u'|u_j)}$ in Eq. 35 (over all triplets (u', u_i, u_j)) for each of these algorithms and found that its numerator is positive, but the denominator is zero. This means that the attacker is certain that this u_j cannot be replaced by this u', and thus these algorithms do not satisfy differential privacy for any positive ϵ.

Runtime. There are two main factors that affect the runtime of our algorithms: the number K of nonsensitive patterns added to U_{c_l} and U'_{c_l}; and the number $|C|$ of all contexts. We examine these in what follows – we do not report results for Baseline, as it is much simpler and thus much faster than our algorithms. We emphasize that the dominant factor in the runtime of our framework is the runtime of the optimization solver, which is what we report below. In contrast, the size of the dataset (length of the input string) only affects the runtime of our framework in the preprocessing stage, which is comparatively much faster.

We first examine K. This parameter determines the size of sets U_{c_l} and U'_{c_l}, which in turn determines the number of constraints and variables in the mathematical programming formulations of our algorithms and hence the runtime of the algorithms. Fig. 9 shows the runtime of each algorithm for varying K. As expected, the runtime of all algorithms increases as K increases. Specifically, the runtime of AQL-MILP is generally higher than that of APG-LP and DP-LP (the runtimes of the latter two algorithms are similar). This is because AQL-MILP is an MILP algorithm (see Section III-D) unlike our other two algorithms which are based on LP, and MILP algorithms are well known to be generally slower than LP algorithms.

We then examine $|C|$, which is equal to the number of times each of our algorithms needs to run to sanitize W. To ensure that we consider all the contexts of sensitive patterns, we varied $|S|$, i.e., the number of sensitive patterns, which in turn determines $|C|$. As can be seen in Fig. 10, the runtime of all algorithms increases with $|C|$. However, our algorithms remain practical, requiring less than three minutes in all tested cases, which include multi-million-letter strings.

Hybrid Algorithms. So far we have demonstrated that each of our algorithms can outperform the others depending



(a) WAPG (b) WAQL (c) WMPG (d) WAQL

(e) WAPG (f) WAQL (g) WAPG (h) WAQL

Fig. 11: APG-MILP: (a) WAPG, (b) WAQL vs. Q_{MAX}; (c) WMPG, (d) WAQL vs. T_{MIN}. DP-MILP: (e) WAPG, (f) WAQL vs. T_{MIN}; (g) WAPG, (h) WAQL vs. ϵ. All results are for the *kasandr* dataset.

on the tested measure. Now we consider whether we can achieve a better tradeoff between privacy gain and quality loss by "combining" two of our algorithms to obtain hybrid algorithms, APG-LP and AQL-MILP to obtain APG-MILP, and AQL-MILP and DP-LP to obtain DP-MILP.

APG-MILP maximizes APG while ensuring that AQL does not exceed Q_{MAX} and that each possible replacement u' gains at least T_{MIN} privacy. Thus, to obtain APG-MILP, we replace $x_{u'}$ with $z_{u'}$ in APG-LP and add the constraints in Eq. 18 to Eq. 24 to it (see Section V in Supplemental Material for details). As APG-MILP imposes both an upper bound Q_{MAX} on AQL and a lower bound T_{MIN} on privacy gain to control its privacy-quality tradeoff, we evaluated the effects of Q_{MAX} and T_{MIN} separately. Figs. 11a, 11b, 11c, and 11d show the results for the *kasandr* dataset; the results for the other datasets are provided in Section V-B of Supplemental Material. In Figs. 11a and 11b APG-MILP achieves the same WAPG as APG-LP but with a smaller WAQL when varying Q_{MAX}. Although AQL-MILP has the lowest WAQL, it offers a higher WAPG. A similar phenomenon is observed in Figs. 11c and 11d. When varying T_{MIN}, APG-MILP even outperforms AQL-MILP in WMPG in some cases, while its WAQL is much better than that of APG-LP. Consequently, APG-MILP inherits the advantages in terms of privacy gain from both APG-LP and AQL-MILP, while incurring a lower quality loss than APG-LP.

DP-MILP is obtained by adding the differentially private constraint of DP-LP (i.e., Eq. 30) to AQL-MILP. That is, DP-MILP enforces two privacy notions: Minimum Privacy Gain (MPG) for each possible replacement u' and ϵ-differential privacy. Thus, we evaluated DP-MILP by varying T_{MIN} and ϵ separately. In Figs. 11e and 11f, DP-MILP always obtains the highest WAPG, without being worse than the worst of AQL-MILP and DP-LP in terms of WAQL. Similar results for varying ϵ are shown in Figs. 11g and 11h.

V. Conclusion

Strings often need to be disseminated in the context of numerous applications after sanitizing the sensitive patterns occurring in them. However, all existing string sanitization algorithms may reveal the location(s) of sensitive patterns in the sanitized string, thus offering no privacy, when attackers are *strategic*, i.e., when they possess background knowledge about statistics of the input string and also know the algorithm that was used to sanitize the string. In this work, we considered for the first time such strategic attackers and proposed a novel framework to counter them. Within this framework, we designed three mathematical programming algorithms that compute optimal string replacement probabilities, offering different tradeoffs between privacy gain and quality loss. Our algorithms are optimal in the sense that there is no algorithm that performs better against an attacker with the same strategic background knowledge of the substring probabilities and of the sanitization algorithm.

Our experimental results show that our three algorithms outperform a natural baseline, offering both higher privacy gain and lower quality loss when one is kept constant compared to the other. Furthermore, we proposed two hybrid, optimal algorithms which provide a better tradeoff between privacy and quality compared to their component algorithms.

The main decision for a practitioner who wishes to integrate our approach with a real-world system is how to model the attacker's background knowledge. The most reasonable approach is to look for publicly available information that the attacker may be aware of. For example, there are publicly available DNA reference human genome sequences (see

Motivation subsection in Section I) or publicly available information about sequences of location visits by people in a city. After constructing a sanitized string Z by any of our algorithms, the practitioner can directly use it in any task in which they would use the original string W. For example, they could index Z (instead of W) to answer pattern matching queries or to discover frequent patterns.

REFERENCES

[1] B. A. Malin and L. Sweeney, "Determining the identifiability of DNA database entries," in *Proc. AMIA Annu. Symp.*, 2000, pp. 537–541.

[2] 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.

[3] B. Berendt and M. Spiliopoulou, "Analysis of navigation behaviour in web sites integrating multiple information systems," *VLDB J.*, vol. 9, no. 1, pp. 56–75, 2000.

[4] S. R. M. Oliveira and O. R. Zaïane, "Protecting sensitive knowledge by data sanitization," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2003, pp. 613–616.

[5] J. J. Ying, W. Lee, T. Weng, and V. S. Tseng, "Semantic trajectory mining for location prediction," in *Proc. ACM SIGSPATIAL Int. Symp. Adv. Geographic Inf. Syst. (ACM-GIS)*, 2011, pp. 34–43.

[6] G. Poulis, S. Skiadopoulos, G. Loukides, and A. Gkoulalas-Divanis, "Apriori-based algorithms for k^m-anonymizing trajectory data," *Trans. Data Priv.*, vol. 7, no. 2, pp. 165–194, 2014.

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

[8] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proc. IEEE Symp. Security Privacy (SP)*, 2008, pp. 111–125.

[9] G. Theodorakopoulos, R. Shokri, C. Troncoso, J. Hubaux, and J. L. Boudec, "Prolonging the hide-and-seek game: Optimal trajectory privacy for location-based services," in *Proc. ACM Workshop Privacy Electron. Soc. (WPES)*, 2014, pp. 73–82.

[10] 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, algorithms, and experiments," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 6, pp. 5948–5963, 2023.

[11] 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," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 1, pp. 8:1–8:34, 2021.

[12] G. Bernardini, H. Chen, A. Conte, R. Grossi, G. Loukides, N. Pisanti, S. P. Pissis, and G. Rosone, "String sanitization: A combinatorial approach," in *Proc. Eur. Conf. Mach. Learn. Knowl. Discov. Databases (ECML PKDD)*, vol. 11906, 2019, pp. 627–644.

[13] G. Bernardini, H. Chen, G. Loukides, N. Pisanti, S. P. Pissis, L. Stougie, and M. Sweering, "String sanitization under edit distance," in *Proc. Annu. Symp. Combin. Pattern Matching (CPM)*, vol. 161, 2020, pp. 7:1–7:14.

[14] A. Gkoulalas-Divanis and G. Loukides, "Revisiting sequential pattern hiding to enhance utility," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining (KDD)*, 2011, pp. 1316–1324.

[15] R. Gwadera, A. Gkoulalas-Divanis, and G. Loukides, "Permutation-based sequential pattern hiding," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2013, pp. 241–250.

[16] O. Abul, F. Bonchi, and F. Giannotti, "Hiding sequential and spatiotemporal patterns," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 12, pp. 1709–1723, 2010.

[17] G. Loukides and R. Gwadera, "Optimal event sequence sanitization," in *Proc. SIAM Int. Conf. Data Mining (SDM)*, 2015, pp. 775–783.

[18] J. Natwichai, X. Li, and M. E. Orlowska, "Hiding classification rules for data sharing with privacy preservation," in *Proc. Int. Conf. Data Warehousing Knowl. Discov. (DaWaK)*, vol. 3589, 2005, pp. 468–477.

[19] A. Gkoulalas-Divanis and V. S. Verykios, "Exact knowledge hiding through database extension," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 5, pp. 699–713, 2009.

[20] 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.

[21] R. Shokri, G. Theodorakopoulos, C. Troncoso, J. Hubaux, and J. L. Boudec, "Protecting location privacy: optimal strategy against localization attacks," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 617–627.

[22] "Human genomic variation," 2024, [Online]. Available: https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation.

[23] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Tech. J.*, vol. 28, no. 4, pp. 656–715, 1949.

[24] R. Shokri, G. Theodorakopoulos, G. Danezis, J. Hubaux, and J. L. Boudec, "Quantifying location privacy: The case of sporadic location exposure," in *Proc. Priv. Enhancing Technol. Symp. (PETS)*, vol. 6794, 2011, pp. 57–76.

[25] R. Shokri, G. Theodorakopoulos, and C. Troncoso, "Privacy games along location traces: A game-theoretic framework for optimizing location privacy," *ACM Trans. Priv. Secur.*, vol. 19, no. 4, pp. 11:1–11:31, 2017.

[26] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. Theory Cryptogr. Conf. (TCC)*, vol. 3876, 2006, pp. 265–284.

[27] Univ. Wisconsin, "Escherichia coli dataset," 2021, [Online]. Available: https://bacteria.ensembl.org/Escherichia_coli_str_k_12_substr_mg1655_gca_000005845/Info/Index/.

[28] D. Heckerman, "MSNBC dataset," [Online]. Available: https://doi.org/10.24432/C5390X.

[29] "Trucks dataset," 2019, [Online]. Available: https://bitbucket.org/stringsanitization/stringsanitizationpkdd19/src/master/truck_char.txt.

[30] S. Sidana, C. Laclau, and M.-R. Amini, "KASANDR dataset," 2017, [Online]. Available: https://doi.org/10.24432/C5PK7M.

[31] D. Sikeridis, I. Papapanagiotou, and M. Devetsikiotis, "IoT dataset," [Online]. Available: https://ieee-dataport.org/open-access/crawdad-unmblebeacon.

[32] 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, pp. 14:1–14:53, 2010.

[33] G. Gürsoy, P. Emani, C. M. Brannon, O. A. Jolanki, A. Harmanci, J. S. Strattan, J. M. Cherry, A. D. Miranker, and M. Gerstein, "Data sanitization to reduce private information leakage from functional genomics," *Cell*, vol. 183, no. 4, pp. 905–917, 2020.

[34] J. C. Lin, J. M. Wu, P. Fournier-Viger, Y. Djenouri, C. Chen, and Y. Zhang, "A sanitization approach to secure shared data in an IoT environment," *IEEE Access*, vol. 7, pp. 25 359–25 368, 2019.

[35] G. Qiu, T. Bai, G. Tang, D. Guo, C. Li, Y. Gan, B. Zhou, and Y. Shen, "Quantifying privacy risks of behavioral semantics in mobile communication services," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 1908–1923, 2025.

[36] J. Kang, D. Steiert, D. Lin, and Y. Fu, "MoveWithMe: Location privacy preservation for smartphone users," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 711–724, 2020.

[37] Z. Zheng, Z. Li, S. Long, S. Guo, C. Chen, and K. Xu, "Pricing utility vs. location privacy: A differentially private data sharing framework for ride-on-demand services," *IEEE Trans. Dependable Secur. Comput.*, vol. 22, no. 4, pp. 3497–3513, 2025.

[38] Y. Jiang, T. Shang, and J. Liu, "Privacy-preserving multiple sequence alignment scheme for long gene sequence," *Proc. Priv. Enhancing Technol.*, vol. 2025, no. 1, pp. 236–249, 2025.

[39] J. M. Wu, G. Srivastava, J. C. Lin, and Q. Teng, "A multi-threshold ant colony system-based sanitization model in shared medical environments," *ACM Trans. Internet Techn.*, vol. 21, no. 2, pp. 49:1–49:26, 2021.

[40] J. Zhang, Z. Tian, M. Zhu, Y. Song, T. Sheng, S. Yang, Q. Du, X. Liu, M. Huang, and D. Li, "DYNTEXT: semantic-aware dynamic text sanitization for privacy-preserving LLM inference," in *Proc. Assoc. Comput. Linguistics (ACL) Findings*, 2025, pp. 20 243–20 255.

[41] T. Yang, X. Zhu, and I. Gurevych, "Robust utility-preserving text anonymization based on large language models," in *Proc. Annu. Meet. Assoc. Comput. Linguistics (ACL)*, 2025, pp. 28 922–28 941.

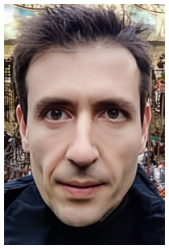
[42] X. Sun and P. S. Yu, "A border-based approach for hiding sensitive frequent itemsets," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2005, pp. 426–433.

[43] P. Krasadakis, G. Futia, V. S. Verykios, and E. Sakkopoulos, "An end-to-end knowledge graph solution to the frequent itemset hiding problem," *Inf. Sci.*, vol. 672, p. 120680, 2024.
[44] Y. Gui, W. Gan, Y. Wu, and P. S. Yu, "Privacy preserving rare itemset mining," *Inf. Sci.*, vol. 662, p. 120262, 2024.
[45] Y. Wu, C. Chiang, and A. L. P. Chen, "Hiding sensitive association rules with limited side effects," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 29–42, 2007.
[46] S. S. Aljehani and Y. Alotaibi, "Preserving privacy in association rule mining using multi-threshold particle swarm optimization," *Inf. Sci.*, vol. 692, p. 121673, 2025.
[47] G. Zhao, D. Chen, and M. Zhang, "Hiding patterns with gaps in sequential data," *Concurr. Comput. Pract. Exp.*, vol. 37, no. 21-22, p. e70187, 2025.
[48] H. Tang, K. Chen, Z. Xie, and C. Wang, "Artificial impostors: An efficient and scalable scheme for location privacy preservation," *IEEE Trans. Serv. Comput.*, vol. 18, no. 3, pp. 1262–1277, 2025.
[49] G. Bernardini, C. Liu, G. Loukides, A. Marchetti-Spaccamela, S. P. Pissis, L. Stougie, and M. Sweering, "Missing value replacement in strings and applications," *Data Min. Knowl. Discov.*, vol. 39, no. 2, p. 12, 2025.
[50] S. Gambs, M. Killijian, and M. N. del Prado Cortez, "Next place prediction using mobility Markov chains," in *Proc. Workshop Measurement, Privacy, Mobility (MPM)*, 2012, pp. 3:1–3:6.
[51] M. Crochemore, C. Hancart, and T. Lecroq, *Algorithms on strings*. Cambridge University Press, 2007.
[52] J. J. Jiang and D. W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," in *Proc. Res. Comput. Linguistics Int. Conf. (ROCLING)*, 1997, pp. 19–33.
[53] J. D. O'Brien, V. N. Minin, and M. A. Suchard, "Learning to count: Robust estimates for labeled distances between molecular sequences," *Molecular Biology and Evolution*, vol. 26, no. 4, pp. 801–814, 01 2009.
[54] M. Alamuri, B. R. Surampudi, and A. Negi, "A survey of distance/similarity measures for categorical data," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2014, pp. 1907–1914.
[55] U. Manber and E. W. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM J. Comput.*, vol. 22, no. 5, pp. 935–948, 1993.
[56] G. Sierksma and Y. Zwols, *Linear and Integer Optimization: Theory and Practice, Third Edition*. CRC press, 2015.
[57] M. Asghari, A. M. Fathollahi-Fard, S. M. J. Mirzapour Al-e hashem, and M. A. Dulebenets, "Transformation and linearization techniques in optimization: A state-of-the-art survey," *Mathematics*, vol. 10, no. 2, 2022.
[58] G. Bernardini, H. Chen, G. Fici, G. Loukides, and S. P. Pissis, "Reverse-safe text indexing," *ACM J. Exp. Algorithmics*, vol. 26, pp. 1.10:1–1.10:26, 2021.
[59] E. Ukkonen, "Approximate string matching with q-grams and maximal matches," *Theor. Comput. Sci.*, vol. 92, no. 1, pp. 191–211, 1992.
[60] J. Lin, "Divergence measures based on the Shannon entropy," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 145–151, 1991.

Pengxin Bian (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree in Computer Science in the Department of Informatics at King's College London. Her research interests focus on data privacy and data mining.

George Theodorakopoulos is a reader (associate professor) with Cardiff University. His research focuses on privacy and security.

Solon P. Pissis is a senior researcher with CWI and an associate professor with the Vrije Universiteit, both in Amsterdam. His research focuses on theory of algorithms and their application in data mining.



Grigorios Loukides (Senior Member, IEEE) is an associate professor with King's College London. His research focuses on data mining.

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org., provided by the authors. This supplementary material includes additional related work, algorithmic details, the proof for linearization of the AQL-MILP algorithm, and additional experimental results to support the main article.