Revisiting Sequential Pattern Hiding to Enhance Utility

Aris Gkoulalas-Divanis
Information Analytics Lab
IBM Research – Zurich
Rüschlikon, Switzerland
agd@zurich.ibm.com

Grigoris Loukides
Health Information Privacy Lab
Vanderbilt University
Nashville, Tennessee, USA
g.loukides@vanderbilt.edu

ABSTRACT

Sequence datasets are encountered in a plethora of applications spanning from web usage analysis to healthcare studies and ubiquitous computing. Disseminating such datasets offers remarkable opportunities for discovering interesting knowledge patterns, but may lead to serious privacy violations if sensitive patterns, such as business secrets, are disclosed. In this work, we consider how to sanitize data to prevent the disclosure of sensitive patterns during sequential pattern mining, while ensuring that the nonsensitive patterns can still be discovered. First, we re-define the problem of sequential pattern hiding to capture the information loss incurred by sanitization in terms of both events’ modification (distortion) and lost nonsensitive knowledge patterns (side-effects). Second, we model sequences as graphs and propose two algorithms to solve the problem by operating on the graphs. The first algorithm attempts to sanitize data with minimal distortion, whereas the second focuses on reducing the side-effects. Extensive experiments show that our algorithms outperform the existing solution in terms of data distortion and side-effects and are more efficient.

Categories and Subject Descriptors
H.2.8 [Database Management]: Database Applications—Data Mining

General Terms
Algorithms, Security, Performance

Keywords
Data privacy, Knowledge hiding, Sequential pattern hiding

1. INTRODUCTION

Sequential data are increasingly collected to support numerous applications in which the sequentiality of events is of primary interest. Examples of such data are web usage logs, which record web page accesses, or mobility data that capture the location of mobile devices at different moments in time [16]. Clearly, sequential data offer remarkable opportunities for discovering interesting behavioral patterns that can be beneficial to a broad community of people. For example, mining user mobility data can reveal interesting patterns that aid traffic engineers and environmentalists in their decisions. Publishing sequential data for data mining purposes, however, may lead to serious privacy violations, if sensitive knowledge patterns are discovered. For instance, the mining of knowledge patterns from mobility datasets may enable intrusive inferences regarding the habits of a portion of the population, or provide the means for unsolicited advertisement and user profiling. Similar concerns have also been raised related to medical data sharing [13,21].

To address these concerns, knowledge hiding methods [5] are necessary. These methods conceal sensitive patterns that can otherwise be mined from published data, without seriously affecting the data and the nonsensitive interesting patterns. Clifton and Marks [12], following D. E. O’Leary [24] who firstly pointed out the privacy breaches that originate from data mining algorithms, indicated the need to consider data mining approaches under the prism of privacy preservation. Since then, several methods emerged to hide knowledge that appears in the form of frequent itemsets and related association rules [19,27,29], or classification rules [11,23].

Unlike these works, this paper considers the problem of hiding sensitive knowledge that appears in the form of frequent sequences and can be disclosed through sequence pattern mining algorithms [6,9]. Sequential pattern hiding is a challenging problem, because sequences have more complex semantics than itemsets, and calls for efficient solutions that offer high utility. To our knowledge, only the work of [3,4] attempts to address this problem, but it may fail to identify high-quality hiding solutions, as we discuss in Section 2.

Our work makes the following contributions:

• We re-define the problem of sequential pattern hiding to capture the utility of released data by considering both the side-effects and the distortion introduced by the hiding process. This allows the production of more useful data for the task they are disseminated for.

• We design two novel sequence hiding algorithms. The first algorithm aims to minimize data distortion, whereas the second focuses on ensuring that the nonsensitive interesting knowledge can still be discovered.

• We extensively evaluate our algorithms, demonstrating that they significantly outperform the existing solution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD ’11, August 21–24, 2011, San Diego, California, USA.
Copyright 2011 ACM 978-1-4503-0813-7/11/08 ...$10.00.
in terms of reducing data distortion and side-effects, while being an order of magnitude more efficient.

The rest of the paper is organized as follows. Related work is in Section 2, where we also discuss the limitations of existing approaches. Section 3 provides the background. In Section 4, we present the hiding strategies used by our algorithms, which are given in Section 5. Section 6 contains the experimental evaluation and Section 7 concludes the paper.

2. RELATED WORK

Approaches for privacy-preserving data sharing fall into two general categories [5]. The first category of approaches attempt to protect the privacy of individuals, whose information is contained in the data, by preventing the disclosure of individuals’ identity [28] or sensitive information [8,10,15]. The second category, referred to as knowledge hiding, aims to prevent sensitive patterns from being mined from the data, and it is the one our work is related to.

Several hiding algorithms have been proposed with most of the research being conducted along the lines of protecting sensitive association and classification rules. In particular, association rule hiding evolved from efficient heuristic approaches [7,25,29], to border-based approaches [27] and, more recently, to exact hiding approaches that offer stronger quality guarantees at the expense of high computational cost [17,18,22]. Classification rule hiding, on the other hand, involved around perturbation-based techniques [11] that reduce the confidence of sensitive rules by modifying the values of attributes that support these rules, and reconstruction-based approaches [23] that reconstruct the dataset by using only records supporting nonsensitive rules.

The problem of sequential pattern hiding was recently investigated in [4], where the focus was on hiding the sensitive knowledge in a way that minimally affects the support of the rest of the sequences in the database. The authors proposed HHA, a sanitization algorithm that operates as follows. First, for each sequence s of the original database, this algorithm computes the different ways (called matchings) in which s supports any sensitive sequence s’. Then, the original sequences are sorted in ascending order with respect to the number of matchings that they contribute to, and the top sequences are selected for sanitization, based on a user-specified disclosure threshold ψ. The sanitization operation eliminates all matches of the sensitive sequences in the sequence s by marking selected events with a special symbol Δ. To sanitize the sequence s, HHA finds, for each event e in s, the number of matchings δ(e) to which e contributes, and marks the events that contribute to most matches, until s no longer supports sensitive sequences. The approach of [4] has three limitations, which we address in this work.

First, the problem formulation adopted by [4] does not focus on side-effects that may be introduced by the hiding process to nonsensitive frequent sequences. As a result, a large number of potentially interesting sequences may be lost in the sanitized dataset. Since the main reason behind publishing sequential data for data mining purposes is to enable the discovery of nonsensitive frequent sequences, the problem formulation of [4] may lead to producing solutions of low data utility. As can be seen in Fig. 1(d), which shows the result of applying the HHA algorithm [4] with ψ = 3 to the dataset of Fig. 1(a), the nonsensitive frequent sequences n1, n2, n3, n4 and n5 are lost. This, however, can be avoided for n1 and n3, as shown in Fig. 1(f).

Second, the (global) selection criterion, used by HHA to identify sequences for sanitization, often selects sequences that incur high distortion when sanitized. That is, it may mark more events than necessary with Δ. Consider, for example, a sensitive sequence s’ = < a, b, c > and the sequences s1 = < a, a, b, c, > and s2 = < a, a, b, b, c, c, >. Both s1 and s2 contain (i.e., support) s’, but s1 does so with 9 matchings, s2 with only 8. As a result, HHA chooses to sanitize s2 instead of s1, assuming that this would incur less distortion. Notice, however, that s1 should be sanitized instead, since the minimum number of events that should be marked with Δ to sanitize s1 is 1 (i.e., b), whereas 2 events need to be marked with Δ to sanitize s2 (i.e., both a’s, or both b’s, or both c’s).

Third, HHA needs to compute δ(e) for each event e in every sequence s. As we discuss in Section 5.2.1, our algorithms avoid this costly operation to improve efficiency.

3. BACKGROUND

In this section, we provide the background for the sequence hiding problem and derive the problem statement. Without loss of generality, we follow the work of [4] in that we consider patterns that are simple sequences of symbols (or events). As we discuss in Section 5.2.4, our algorithms can easily be extended to support sequential patterns that follow the classical definition proposed in [6].

Let \( \Sigma = \{ e_1, e_2, \ldots, e_M \} \) be a set of literals \( e_i \), called symbols or events, where \( M \) denotes the cardinality of the set. A sequence \( s \) is an ordered list of events from \( \Sigma \) and is represented as \( s = < e_{i_1}, e_{i_2}, \ldots, e_{i_m} > \), where \( e_i \in \Sigma \), \( \forall i \in \{1, \ldots, m\} \). A transaction \( T \) is an ordered pair \((tid, s)\), where \( s \) is a sequence and \( tid \) a unique identifier used to distinguish among transactions that correspond to the same sequence. Furthermore, a sequence database \( D \) is a set of \( N \) transactions, carrying different identifiers. We define the length or size of a sequence \( s \) (and denote it by \( |s| \)) to be the number of events that the sequence contains. Moreover,
we say that a sequence $s_1$ over $\Sigma$ is contained in (or, equivalently, is a subsequence of) a sequence $s_2$ over $\Sigma$, if $s_1$ can be obtained by deleting $|s_2| - |s_1|$ events from sequence $s_2$. We use the symbol "$\subseteq\$" to denote this type of relationship between two sequences, i.e. $s_1 \subseteq s_2$.

A transaction $T = (tid, s)$ supports a sequence $s'$ if $s' \subseteq s$. The support of a sequence $s$ in database $D$ is denoted by $\text{sup}(s, D)$, and it is the number of transactions in $D$ that support $s$. These transactions are called the supporting transactions of sequence $s$. The sequence $s$ is called large or frequent in database $D$, if and only if its support in $D$ is at least equal to a minimum support threshold $\psi$, i.e. $\text{sup}(s, D) \geq \psi$. Otherwise, $s$ is called infrequent in $D$.

The problem of mining sequential patterns requires, given a sequence database $D$ and a minimum support threshold $\psi$, to compute the set of all sequences that are supported by at least $\psi$ transactions in $D$, which is denoted with $\mathcal{F}_D, \psi$. We also formally define the sequence hiding problem below:

**Problem 3.1. (Sequence Hiding Problem)** Given a sequence database $D$, a minimum support threshold $\psi$, and a set of sensitive sequences $\mathcal{S}$ (selected by data owners among the frequent ones), construct a new, sanitized database $D'$ from $D$ such that: (1) $\text{sup}(s, D') \leq \psi$, for each sensitive sequence $s \in \mathcal{S}$, (2) $\mathcal{F}_{D', \psi} = \mathcal{F}_{D, \psi} - \mathcal{S}'$, where $\mathcal{S}' = \{s' \in \mathcal{F}_{D', \psi} | s \subseteq s', s \notin \mathcal{S}\}$ and (3) $\text{distance}(D, D')$, the number of events from $D$ that are sanitized in $D'$ is minimized.

The sequence hiding problem requires sanitizing the database $D$ so that (1) no sensitive sequence can be mined from $D'$ at a support threshold of $\psi$ or higher, (2) no side-effects are introduced by the hiding process to $D'$ in terms of (a) sequences in $\mathcal{F}_{D', \psi} - \mathcal{S}'$ that are infrequent in $D'$ or (b) infrequent sequences in $D$ appearing as frequent in $D'$ (ghost sequences), and (3) the least number of events in sequences supported in $D$ is sanitized (marked or deleted) to derive $D'$, which implies that $D'$ should be kept as similar as possible to $D$. If symbols are marked with $\Delta$ when sanitized, then $\text{distance}(D, D')$ is equal to the number of $\Delta$s in $D'$.

As is proved in [4], the problem of sanitizing a sequence by introducing the least distance is NP-hard. In addition, it is important to observe that the three goals in Problem 3.1 are not of the same importance. That is, a sequence hiding algorithm has to accomplish the first goal to protect all the sensitive knowledge, but needs to prioritize between the second and the third goals, when generating a sanitized database. As we will see shortly, the two algorithms we propose guarantee that the first goal will be satisfied, but each focuses on one of the two other goals. Our first algorithm aims to minimize $\text{distance}(D, D')$ to incur low distortion, while the second algorithm favors fewer side-effects instead of a smaller distance, a choice adopted by related work in association rule hiding [19]. We also apply only deletions of events to sanitize the sensitive sequences. Thus, no ghost sequences can be introduced by our approach and the second goal can be re-stated as “minimize the number of non-sensitive frequent sequences that are lost in $D'$”. Moreover, $\text{distance}(D, D') = |\sum_{T \in D, \text{sup}(T, s)}|s| - |\sum_{T \in D', \text{sup}(T, s')}|s'||$.

### 4. HIDING STRATEGIES

In this section, we explain how to represent the matchings between sensitive sequences and transactions using multi-partite graphs (Section 4.1) and how to hide sequences guided by these graphs (Section 4.2). Strategies to perform these operations in a way that reduces distance and helps prevent side-effects are also discussed (Section 4.3).

#### 4.1 Graph-based sequence representation

Let $\mathcal{S}$ be the set of sensitive sequences from $D$, $D_S$ the transactions in $D$ that support at least one sensitive sequence, and $D_D = D - D_S$. A nonsensitive frequent sequence is any sequence in $\mathcal{F}_{D_D, \psi}$, as defined in Problem 3.1. Given a transaction $T$, we use $T_S$ to refer to the set of sensitive sequences $S \in \mathcal{S}$ that $T$ supports in $D$, and $T_N$ to refer to the set of nonsensitive frequent sequences that $T$ supports in $D$. Given a sequence $s = <e_1, e_2, \ldots, e_m>$, we define the position of an event in the sequence: the first event ($e_1$) is in position 0, the last event ($e_m$) is in position $(m - 1)$. For notational convenience, we refer to a transaction $T = (tid, s)$ as $T_{tid} = s$.

**Definition 4.1.** Given a transaction $T \in D$ and a sensitive sequence $s$, we define the matching graph $G_{s, T}$ to be a multi-partite graph with the following properties: (a) each position $i$ of $s$ corresponds to a different part (or layer) $i$ in the graph that carries the symbol of $s$ in this position as a label – the layers are positioned consecutively in $G_{s, T}$ in the same order as the symbols of $s$, (b) for each position in $T$, a node lies in each layer of the graph that has the same label as the corresponding symbol in $T$ – the node carries a label equal to the position of the corresponding symbol in $T$.

Figure 2: Example of (a) constructing graphs for $s, s'$ and a transaction $T$, (b) constructing a single graph for $\{s, s'\}$ and $T$, (c) an original dataset, (d) applying Annotate, (e) avoiding side-effects.
(c) an edge between two nodes $u, v$ exists in the graph only if (i) $u, v$ are in adjacent layers, with $u$ preceding $v$ in the graph, and (ii) $u, v$ belong to a path defined as a sequence of nodes $u_1, u_2, \ldots, u_s$ (based on the corresponding sequence of events in $T$) for which each node $u_i$ is in the $i$-th layer of the graph and $\text{label}(u_1) < \text{label}(u_2) < \ldots < \text{label}(u_s)$—this is called a complete path$^2$, and (d) isolated nodes are removed from $G_{s,T}$.

Definition 4.1 assumes that $|s| > 1$ when $|s| = 1$, $G_{s,T}$ contains one node that has a self-loop, forming a complete path by itself. Figure 2(a) presents an example of constructing the matching graph for a transaction $T$ and a sensitive sequence $s$. Observe that if we assume that, when visited, a node “emits” the symbol of the label of the layer, then traversing a complete path in $G_{s,T}$ corresponds to one occurrence of the sensitive sequence in $T$. Likewise, the number of complete paths in $G_{s,T}$ represents the different ways that the sensitive sequence is “produced” by $T$. To sanitize $T$, all these different matchings have to be removed, which is equivalent to cutting all complete paths in $G_{s,T}$. However, computing the complete paths in the matching graph incurs significant cost [4]. Our algorithms avoid this cost by requiring only the computation of complete paths that pass through a small number of selected nodes, as we show in Section 5. Next, we discuss a graph merging operation that is useful when hiding many sensitive sequences.

**Definition 4.2.** Given a transaction $T \in \mathcal{D}$, a set of sensitive sequences $S \subseteq S$ and the matching graph $G_{s,T}$, for each $s_i \in S$, we construct a matching graph $G_{S,T}$ for $S$ and $T$ by (a) identifying the largest set of nodes $V$ that comprise a single layer in each $G_{s_i,T}$ (breaking ties arbitrarily), (b) for each node in $V$, constructing a node in $G_{S,T}$ that has the same label, (c) for each node that is not contained in $V$, constructing a new node in $G_{S,T}$ that has the same label, and (d) for each edge $(u,v)$ in each $G_{s_i,T}$, creating the edge $(u,v)$ in $G_{S,T}$ and annotating it with the identifier $i$ of $s_i$.

Constructing the matching graph for a set of sensitive sequences allows our algorithms to find solutions of high quality, as we discuss later. Consider that we want to construct a matching graph $G_{S,T}$ for the set $S = \{s, s'\}$ and the transaction $T$ in Fig. 2(a). The largest set of nodes, which comprise a single layer (i.e., nodes that are contained in a single layer that does not have any other nodes) in both $G_{s,T}$ and $G_{s',T}$, contains the nodes labeled 0 and 1. Thus, for each of the latter nodes, we construct a node in $G_{\{s,s'\},T}$ (i.e., we merge these nodes in $G_{s,T}$ and $G_{s',T}$). Next, we construct a node in $G_{\{s,s'\},T}$ for each node that is not labeled 0 or 1 and is contained in $G_{s,T}$ or $G_{s',T}$, and copy the label of the latter node to the new one. For the edge $(0,5)$ in $G_{\{s,s'\},T}$, we create an edge $(0,5)$ in $G_{\{s,s'\},T}$ and annotate this edge as corresponding to $s'$. Last, we perform the same for each other edge in $G_{s,T}$ and also for $G_{s',T}$ to obtain the matching graph $G_{S,T}$ of Fig. 2(b).

### 4.2 Sequence hiding

Matching graphs allow tackling the problem of sequence hiding based on a series of graph operations that guide the

---

2We use “<”, interpreted as “less than”, when comparing the labels of two nodes in the graph. In Fig. 2(a), for instance, label(1) < label(3), since $b$ in the first position of $T$ appears before $c$ in the third position.

sanitization of the transactions of $\mathcal{D}$. Specifically, given one transaction $T$ and a set of sensitive sequences $S$, the sanitization task for $T$ becomes equivalent to applying the minimum number of node deletions in graph $G_{S,T}$, so that all complete paths for the sensitive sequences $s_i \in S$ are cut. As an example, consider the case of sanitizing $T$ so that it no longer supports the sensitive sequence $s_1$, shown in Fig. 2(a). This can be achieved by deleting nodes 0 and 1, or 3, or 4 and 5, or 6. Assuming that the criterion we want to optimize is distance (goal 3 in Problem 3.1), deleting any of nodes 3, or 6 will achieve that with a cost of 1.

An important observation follows if we want to sanitize $T$ so that it no longer supports both sensitive sequences, $s$ and $s'$. If the sanitization is performed separately for $s$ and for $s'$, the hiding algorithm may choose to delete node 3 first, so that $s$ is unsupported by $T$, and then nodes 0 and 1, so that $s'$ is unsupported by $T$. This leads to a total distance of 3, and can be avoided if the hiding algorithm operates on the combined graph $G_{\{s,s'\},T}$, shown in Fig. 2(b). In this case, nodes 0 and 1 participate in all complete paths for both $s$ and $s'$, and their removal can lead to the sanitization of $T$ incurring a distance of 2 instead of 3.

Another issue that needs to be addressed is the update of the matching graph after a delete operation. This is accomplished through a function $\text{Update-path}$ which, given the transaction $T_{id} = s$ and the position $p$ in $s$ in which the symbol to be deleted lies, removes the corresponding node from the matching graph and then traverses the graph to delete any edges and nodes that belong to incomplete paths.

### 4.3 Utility considerations

The hiding of sensitive sequences in $\mathcal{D}$ should be performed in a way that preserves the utility of the database $\mathcal{D}'$ to the highest possible extent. Thus, in this section, we discuss novel strategies that are used by our hiding algorithms to tackle objectives (2) and (3) of Problem 3.1.

**Minimizing distance** Minimizing distance requires finding the minimum number of events that need to be deleted from a transaction, but this is NP-hard [4]. To deal with this problem, we present $\text{Cost}$, which is explained in Algorithm 1. $\text{Cost}$ begins by iterating over all distinct events contained in at least one sequence in $T_S$ (step 1). For each distinct event $e$, it constructs the set of sensitive sequences $S_e$ that contain $e$ and assigns the number of occurrences of $e$ to $w_e$ (steps 2-3). Then, $\text{Cost}$ computes the number of events that need to be deleted from $T$ to sanitize it (steps 4-9). Specifically, it greedily selects the event $e$ with the smallest ratio between $w_e$ and $|S_e|$, and assigns its weight to $w$ (step 7). Next, $\text{Cost}$ adds $w$ to $sc$, which is used to count the number of events to be deleted, and $S_e$ to a set $S$ (steps 8-9). This process is repeated until $S$ contains all sequences in $T_S$. Observe that deleting all occurrences of the selected events from $T$ guarantees that $T$ will be sanitized, because eventually $S = T_S$. Despite its simplicity, $\text{Cost}$ obtains a tight approximation error of $O(\ln(|T_S|))$, as proved in [14].

**Example 1.** Consider applying $\text{Cost}$ on $T_1$ of Fig. 1(a), which supports $s_1$ and $s_4$ of Fig. 1(b). $\text{Cost}$ begins by computing $S_{d} = \{s_1\}$, $w_{d} = 1$, $S_{f} = \{s_4\}$, $w_{f} = 1$, and $S_{b} = \{s_3, s_4\}$, $w_{b} = 3$. Next, it assigns $w_{d}$ and then $w_{f}$ to $w$ as $\frac{w_{d}}{S_{d}} = \frac{1}{1} = 1 < \frac{w_{f}}{S_{f}} = \frac{1}{1} = 1$, and returns $sc = 2$.

**Minimizing side-effects** We present a mechanism that detects events whose deletion does not lead to side-effects.
Algorithm 1: \( \text{Cost}(T, T_S) \)

1. for each (distinct event \( e \) in a sequence in \( T_S \))
2. \( S_e \leftarrow \) the sequences in \( T_S \) that contain \( e \)
3. \( w_e \leftarrow \) the number of occurrences of \( e \) in \( T \)
4. \( \text{sc} \leftarrow 0 \)
5. \( S \leftarrow \emptyset \)
6. while (there is a sensitive sequence \( s \in T_S \setminus S \))
7. \( w \leftarrow \) find the weight \( w_e \) of the event \( e \) with \( \min(\frac{w_e}{w_s}) \)
8. \( \text{sc} \leftarrow \text{sc} + w \)
9. \( S \leftarrow S \cup S_e \)
10. return \( \text{sc} \)

Before doing so, we classify positions of \( T \) as follows. A position \( p \) is annotated as (i) black, if side-effects are not introduced by removing the \( p \)-th event of \( T \), (ii) red if removing \( p \) generates side-effects, or (iii) gray, if we cannot be certain whether the removal of \( p \) will lead to side-effects.

The annotation of \( p \) is performed by \textit{Annotate}, explained in Algorithm 2. After assigning the sequences in \( T_N \) that contain the \( p \)-th event of \( T \) to \( T_{N_p} \), \textit{Annotate} checks whether these sequences have a support different than \( \psi \) in \( D \), and, in this case, annotates \( p \) as black (steps 1-3). If there exist sequences with a support of \( \psi \), \textit{Annotate} iterates over them and annotates \( p \) as red, if at least one of them becomes infrequent after deleting the \( p \)-the event from \( T \) (steps 5-7).

If \( p \) has not been marked red after examining all sequences in \( T_{N_p} \), \textit{Annotate} marks it as gray and returns (step 8).

\textbf{Example 2.} Consider applying \textit{Annotate} to the position 0 of \( T_3 \) in Fig. 2(c) using \( \psi = 1 \). Observe in Fig. 2(d) that the node labeled 0 in \( G_{s;T_3} \), where \( s = \langle d, f, g \rangle \), is annotated black, because the sequences \( n_1 \) to \( n_6 \) are supported by at least 2 transactions. Had we used \( \psi = 2 \) instead, this position would be annotated red, since the sequences \( n_2 \) to \( n_4 \) have a support of 2, and \( n_4, n_5, \) and \( n_6 \) will be lost if the 0-th event is deleted from \( T_3 \).

As explained above, nonsensitive frequent sequences that contain the \( p \)-th event of a transaction \( T \) will be lost if a red position \( p \) is deleted. However, this can be avoided if we locate a set of positions in the graph \( G_{s;T} \) such that (i) they are all annotated black, and (ii) each complete path in \( G_{s;T} \) that contains the node labeled \( p \) also contains a node labeled with at least one of these positions. We are interested in sets containing positions that appear either before or after \( p \) in \( T \). These types of sets are referred to as PA (Preceding Alternatives) and SA (Succeeding Alternatives), respectively. Unfortunately, finding such sets for a position \( p \) incurs significant computational overhead, as it may require examining a very large number of positions. To cope with this overhead, we restrict ourselves to considering positions that are reachable from \( p \) by following at most \( r \) edges in \( G_{s;T} \), where (radius) \( r \in [1, \max_{j=1, \ldots, |s|} |s_j|] \) is a parameter that controls the part of \( G_{s;T} \) to be searched.

Specifically, we find a set PA using \( C-PA(p, G_{s;T}, r) \), a function that traverses \( G_{s;T} \) to find a set of positions that satisfy properties (i) and (ii) mentioned above, lie in layers that appear before the layer of node \( p \) in \( G_{s;T} \), and are reachable from \( p \) by traversing at most \( r \) edges. \( C-PA \) starts with positions that are reachable from \( p \) by following one edge in \( G_{s;T} \) and increases the number of edges that are followed by one in each subsequent iteration. This function stops the first time a set PA is found or when the number of edges that are followed exceeds \( r \). A set SA for \( p \) is found using a function similar to \( C-PA \), called \( C-SA \).

To find sets of positions in a way that helps reduce both distance and side-effects, we use \textit{Replace}, a function that, given a position \( p \) of \( T \), the matching graph \( G_{s;T} \), and a specified \( r \), utilizes \( C-PA \) and \( C-SA \) and returns the smallest of the sets constructed by them.

\textbf{Example 3.} Consider position 3 in \( T_3 \), shown in Fig. 2(c), which corresponds to the node labeled 3 in the matching graph \( G_{s;T} \), and is annotated red. Applying \textit{Replace} to this position using \( r = 1 \) yields \{4, 5, 6\}, which implies that the 4-th, 5-th, and 6-th events of \( T \) can be deleted instead of the 3-rd one. Observe that these positions are annotated black and \( s \) is hidden if their corresponding events are deleted. Applying \( r = 2 \) instead yields \{0, 1\}, since the nodes with these labels are reachable from the node labeled 3 by following two edges, and the positions they correspond to are fewer than those returned by \( C-SA(3, G_{s;T}, 2) \).

Algorithm 2: \( \text{Annotate}(p, D, T_N, \psi) \)

1. \( T_{N_p} \leftarrow \) the nonsensitive frequent sequences in \( T_N \) that contain the \( p \)-th event of \( T \)
2. if (no sequence \( n_j \in T_{N_p} \) has \( \text{sup}(n_j, D) = \psi \))
3. return \( \text{black} \)
4. else
5. for each \( n_j \in T_{N_p} \) such that \( \text{sup}(n_j, D) = \psi \)
6. if \( \text{sup}(n_j, D) < \psi \) after deleting the \( p \)-th event of \( T \)
7. return \( \text{red} \)
8. return \( \text{gray} \)

5. ALGORITHMS

Sequence hiding involves selecting a number of transactions to sanitize and then deciding which events need to be deleted to perform the sanitization. In this section, we discuss our strategies for these steps and present two novel algorithms to perform the hiding task.

5.1 Selecting transactions to sanitize

As our goal is to release a database \( D' \) in which all sensitive sequences are hidden, we first need to select the transactions from \( D \) that will be sanitized. Clearly, all transactions that do not support any sensitive sequence in \( S \) can safely be disclosed and will be part of \( D' \). Among the transactions supporting sensitive sequences, a set of transactions can be chosen to be part of \( D' \) without being sanitized, as long as the support of each sensitive sequence in \( D' \) remains below \( \psi \). This allows the selection of transactions that support sensitive sequences and incur low distortion when sanitized.

To achieve this, we propose \textit{Exclude}, a function that is used by our sanitization algorithms and described in Algorithm 3. \textit{Exclude} begins by computing the number of deletions required to sanitize each transaction in \( D_S \) using \textit{Cost}, and then sorts these transactions in descending order with respect to this cost (steps 1-3). Next, it iterates over each transaction \( T \) in \( D_S \) and moves the transaction to \( D' \) without sanitizing it only if every sensitive sequence supported by \( T \) has been encountered fewer than \((\psi - 1) \) times. The remaining transactions in \( D_S \) are subsequently sanitized.

5.2 Sanitizing sequences

In what follows, we introduce \( DBSH \), an algorithm that employs \textit{Exclude} and \textit{Cost} to minimize distance, and \( SBHS \) which reduces side-effects by utilizing \textit{Replace}. The differences between these two algorithms and their extension to support itemset sequences, are also discussed.

1320
5.2.1 Distance Based Sequence Hiding (DBSH)

DBSH is applied to $D_S$ and its pseudocode is provided in Algorithm 4. It first uses Exclude to create a potentially smaller set of transactions, sorted in terms of sanitization cost and assigned to $D_S$ (step 1), and then selects a transaction $T$ from $D_S$ to sanitize (step 2). In step 3, DBSH uses a function Candidates to find an approximately minimal set of positions of $T$ corresponding to events that need to be deleted to sanitize $T$. This function is similar to Cost except that it returns a set $\{P, H\}$, where $P$ is a 2D array and $H$ a hashtable. Each element $P[j]$ of $P$ contains the positions of $T$ that need to be deleted to hide a subset $S_j \in T_S$, comprised of sequences that share at least one event, and it is associated to $S_j$ using $H$. In addition, $P$ is sorted in descending order of the size of the subsets of sequences its elements correspond to. Next, in steps 4-5, DBSH retrieves the element $P[j]$ of $P$ and the subset $S_j \in T_S$ corresponding to $P[j]$, using $H$. Following that, it iteratively deletes events from the positions of $T$ that are retrieved from $P[j]$ as we increment $i$ by 1, if $P[j][i]$ corresponds to a valid position, until $T$ no longer supports $S_j$ (steps 7-13). This process is repeated for each $S_j \in T_S$ and all transactions of $D_S$.

**Algorithm 3** Exclude($D_S, \psi$)
1. for each (transaction $T$ in $D_S$)
2. compute Cost($T, T_S$)
3. sort transactions in $D_S$ in descending order of cost
4. for each (transaction $T$ in $D_S$)
5. $M \leftarrow$ a hashtable associating each $s \in T_S$ with 0
6. if ($s$ < $\psi - 1$), for each $s \in T_S$)
7. $D_S \leftarrow D_S \setminus T$
8. for each ($s \in T_S$)
9. increase $M(s)$ by 1

**Algorithm 4** DBSH($D_S, \psi$)
1. $D_S \leftarrow$ Exclude($D_S, \psi$)
2. for each (transaction $T$ in $D_S$)
3. $P = \{P, H\} \leftarrow$ Candidates($T, T_S$)
4. for each (element $P[j]$ in $P$)
5. $S_j \leftarrow H(j)$ (the subset of $T_S$ corresponding to $P[j]$)
6. $i \leftarrow 0$
7. while ($T$ supports $S_j$)
8. delete the event in the $P[j][i]$-th position of $T$
9. Update-paths($p, T$)
10. if ($T$ supports $S_j$)
11. if ($P[j][i] = \psi$) < $P[j][i]$ not a valid position
12. break
13. $i \leftarrow i + 1$
14. return $D_S$

**Cost analysis** DBSH needs $O(\vert D \vert \times \vert T \vert^2 \times \vert s \vert)$ time in the worst case when all transactions in $D$ support at least one sequence in $S$, sensitive sequences do not share events, and $\psi = 1$, where $|s|$ is the length of the largest sequence in $S$. The bottleneck of DBSH is Update-paths, which takes $O(|T| \times |s|)$ time [4].

5.2.2 Side-effect Based Sequence Hiding (SBSH)

SBSH is explained in Algorithm 5. This algorithm begins by calling Exclude-SBSH, a variant of Algorithm 3 that removes the transaction that supports the largest number of sensitive sequences from $D_S$, breaking ties arbitrarily if many transactions have the same sanitization cost. This choice can significantly reduce the number of side-effects incurred by sanitization, as our experiments confirm. Then, SBSH considers the first transaction in $D_S$ and uses Candidates-SBSH, which is similar to Candidates except that it creates an 1D array $P$ containing positions that correspond to all sequences of $T_S$ (steps 2-3). After that, in step 4, SBSH uses a variant of Annotate, which annotates every position corresponding to an event contained in a sensitive sequence $s \in T_S$ and stores the result for each position into an array $A$. In steps 7-37, SBSH retrieves the element $P[i]$ of $P$ and selects a set of events that can be deleted to sanitize $T$ in such a way that side-effects are prevented. Note that the support of sequences in $T_N$ will not reflect whether a side-effect has occurred if a red position is marked for deletion but has not yet been deleted from $T$. To detect side-effects accurately, we compute supports as if the marked nodes had been removed and use a function Max-updated-support in step 7, which returns the maximum support of sequences in $T_S$.

**Algorithm 5** SBSH($D_S, \psi, r, D_N$)
1. $D_S \leftarrow$ Exclude-SBSH($D_S, \psi$)
2. for each (transaction $T$ in $D_S$)
3. $P = \{P, H\} \leftarrow$ Candidates-SBSH($T, T_S$)
4. $A \leftarrow$ Annotate($T, D, T_N, \psi$)
5. $R \leftarrow \emptyset$ (positions of events to delete from $T$
6. $i \leftarrow 0$
7. while($\text{Max-updated-support}(T, T_S) \geq \psi$)
8. if ($A[P[i]]$ is black)
9. $R \leftarrow R \cup P[i]$
10. Update-paths($P[i], T$)
11. if ($P[i] = \psi$) < $P[i]$ is not a valid position
12. break
13. else
14. if ($A[P[i]]$ is red)
15. $R' \leftarrow \emptyset$ (alternative positions to $P[i]$)
16. for each ($S_j \leftarrow H(j)$)
17. if ($G_{S_j, T}$ has a path that contains only $P[i]$)
18. break
19. else
20. $R' \leftarrow R' \cup \text{Replace}(P[i], G_{S_j, T}, r)$
21. if ($R' = \emptyset$)
22. $R \leftarrow R \cup P[i]$
23. Update-paths($P[i], T$)
24. if ($P[i] = \psi$)
25. break
26. $i \leftarrow i + 1$
27. else
28. for each ($p' \in R'$)
29. $R' \leftarrow R' \cup \{p'\}
30. Update-paths($P[i], T$)
31. else
32. if ($\text{Annotate}(P[i], D, T_N, \psi)$ is gray)
33. $R \leftarrow R \cup P[i]$
34. Update-paths($P[i], T$)
35. if ($P[i] = \psi$)
36. break
37. $i \leftarrow i + 1$
38. for each ($p \in R$)
39. delete $p$-th event of $T$
40. return $D_S$

Contrary to DBSH, the SBSH algorithm detects the events to delete based on the annotation of their positions. If event $P[i]$ is annotated black, the $P[i]$-th event of $T$ can be deleted from $T$ without introducing a side-effect. Thus, we mark $P[i]$ as a position of an event to be deleted by adding it into $R$, update $G_{S_j, T}$ using Update-paths to reflect this marking, and increment $i$ by 1. If $P[i]$ corresponds to a valid position (steps 9-13), if $P[i]$ is annotated red, we first iterate over each subset of sequences $S_j \in T_S$ to identify alternative positions for $P[i]$ (steps 15-20). Such positions are found by applying Replace with the specified radius $r$, and they are added to $R'$, only if $G_{S_j, T}$ contains a path that has $P[i]$ and at least another node, as $P[i]$ needs to be removed from $T$ in any other case. Note that we replace a single position $p$ at a time and use a function $\text{Max-updated-support}$ in step 7, which returns the maximum support of sequences in $T_S$. We use a function $\text{Max-updated-support}$ to create a potentially smaller set of transactions, sorted in terms of sanitization cost and assigned to $D_S$, breaking ties arbitrarily if many transactions have the same sanitization cost. This choice can significantly reduce the number of side-effects incurred by sanitization, as our experiments confirm. Then, SBSH considers the first transaction in $D_S$ and uses Candidates-SBSH, which is similar to Candidates except that it creates an 1D array $P$ containing positions that correspond to all sequences of $T_S$ (steps 2-3). After that, in step 4, SBSH uses a variant of Annotate, which annotates every position corresponding to an event contained in a sensitive sequence $s \in T_S$ and stores the result for each position into an array $A$. In steps 7-37, SBSH retrieves the element $P[i]$ of $P$ and selects a set of events that can be deleted to sanitize $T$ in such a way that side-effects are prevented. Note that the support of sequences in $T_N$ will not reflect whether a side-effect has occurred if a red position is marked for deletion but has not yet been deleted from $T$. To detect side-effects accurately, we compute supports as if the marked nodes had been removed and use a function $\text{Max-updated-support}$ in step 7, which returns the maximum support of sequences in $T_S$.
5.2.3 DBSH and SBSH through an example

Consider applying DBSH and SBSH to the data of Fig. 1(a) and the sensitive sequences of Fig. 1(b) using $\psi = 3$ and, for SBSH, $r = 3$. The sanitized datasets are shown in Figs. 1(e) and 1(f), respectively. Observe that DBSH removed the 0-th event of $T_1$, while SBSH removed the 0-th event of $T_2$. Although both $T_1$ and $T_2$ have a sanitization cost of 1, as can be seen in Fig. 4(a), $T_2$ supports fewer nonsensitive frequent sequences. Since this implies potentially fewer side-effects, SBSH selects this transaction instead of $T_1$. Similarly, DBSH sanitizes $T_2$, while SBSH chooses to sanitize $T_1$ instead. Note that, due to these choices, SBSH can produce a significantly less distorted sanitized database than both HHA and DBSH with respect to distance and side-effects, as shown in Fig. 4(b).

5.2.4 Supporting itemset sequences

Our algorithms can easily be extended to operate on itemset sequences [6], where each event $e$ is an itemset instead of a symbol (item). In this case, to construct the matching graph for a transaction $T$ and a sensitive sequence $s$, instead of applying equality tests between the events of $T$ ($e_i$) and $s$ ($e_j$), we apply set inclusion tests (i.e., $e_i \subseteq e_j$). Moreover, to sanitize an event $e_i$ in $T$, we choose to delete the subset of items in $e_i$ that affect most of the complete paths in $G_{s,T}$. We compare DBSH and SBSH against HHA [4], in terms of data utility and efficiency. We capture utility by measuring distance and number of side-effects. For this purpose, we mined sequential patterns using PrefixSpan [26] with different support thresholds and considered a varying number of sensitive sequences of different lengths. We use the notation $\Psi_l$ to refer to $\Psi$ sequences of length $l$. To ensure a fair comparison, we configured SBSH with $\tau = 1$, unless stated otherwise. We used three real-world datasets: MSNBC (MSN) [20], BMS-WebView-1 (BMS) [1], and TRUCKS [2], which have been used in evaluating relevant works [3,4]. The characteristics of these datasets and the minimum support thresholds used are shown in Table 1 (default values appear in bold).

| Dataset | $N$ | $|\mathcal{T}|$ | $\text{Avg. } |\mathcal{T}|$ | min. sup. threshold $\psi$ |
|---------|-----|----------------|-----------------|------------------|
| MSN     | 980198 | 17 | 5.7 | 3711.4949898.19796 |
| BMS     | 59902 | 495 | 2.5 | 59 |
| TRUCKS  | 275 | 100 | 20.1 | 15, 20, 30, 40 |

Table 1: Characteristics of datasets and $\psi$ values used.
6.1 Side-effects

We first demonstrate that our hiding algorithms allow potentially interesting nonsensitive patterns to be discovered. The results with respect to the percentage of side-effects incurred when hiding increasingly larger sets of patterns of varying length are illustrated in Fig. 3. Observe that SBSH produced sanitized data that permit the mining of at least 12% and up to 23% more nonsensitive patterns than HHA, which incurred 26.4 times more side-effects on average. DBSH outperformed HHA but not SBSH, since it does not attempt to reduce side-effects. To illustrate the effect of varying the length of sensitive patterns more clearly, we provide Fig. 5(a). Observe that our methods consistently outperformed HHA, which incurred at least 2 times more side-effects than SBSH.

Next, we show that SBSH can reduce side-effects when applied with a large $r$. For space reasons, we report results only for the TRUCKS dataset, which is the most difficult to sanitize owing to its large density, and for two configurations of SBSH, with the minimum and maximum $r$. Figure 5(b) demonstrates a scenario in which an increasing number of sensitive sequences need to be hidden. Using a large $r$ helps avoid side-effects, reducing their number by 27 on average. This is also observed in Fig. 5(c), which shows the impact of hiding sensitive patterns that vary in length. Also, increasing $r$ helps hide large numbers of sensitive patterns because SBSH operates on larger graphs.

6.2 Distance

We examined the impact of hiding patterns of varying number and length on distance. Figure 6 reports the percentage of events deleted by each of the methods tested. Note that HHA removed 14 and 15.4 times more events than SBSH and DBSH on average, and SBSH was only slightly worse than DBSH, although focuses on side-effects. We also provide Fig. 8(a), which shows that our algorithms outperformed HHA when long patterns need to be hidden.

We then studied the impact of $r$ on distance by executing SBSH with the minimum and the maximum $r$ value. Figure 8(b) shows the result when an increasing number of patterns are hidden, and Fig. 8(c) when the sensitive patterns vary in length. We observed that $r$ had little impact on distance, which implies that SBSH can find solutions with fewer side-effects without overly distorting data.
6.3 Efficiency

We also tested the time each method needs to conceal the sensitive patterns. Figure 7(a) reports this for the case of hiding various patterns in MSN. Observe that our algorithms were at least an order of magnitude faster than HHA, scaling linearly with the number of sensitive patterns, and not exponentially as HHA did. SBSH was slower than DBSH by 44% on average, owing to the overhead that processing nonsensitive patterns incurs. Similar results were observed for BMS, as can be seen in Fig. 7(b).

We then measured runtime as a function of the minimum support thresholds $\psi$. For consistency, the patterns were mined and hidden using the same $\psi$, which varied from 3711 (0.375%) to 19796 (2%). The result for hiding 50 patterns of length 3 from MSN is shown in Fig. 7(c). All algorithms needed more time when $\psi$ was smaller, because there are more nonsensitive sequences that need to be dealt with. However, DBSH and SBSH remained faster than HHA by at least 5.5 and 4.6 times, respectively.

Last, we investigated the effect of using a larger $r$ in SBSH on the runtime. Figures 9(a), 9(b), and 9(c) show that SBSH remained efficient, even when it ran on MSN with the maximum $r$ value. In fact, SBSH needed less than 13% more time on average to search a larger part of the matching graphs. The runtime of SBSH was also no more than 41% larger (on average) when it ran on the denser TRUCKS dataset, as shown in Figs. 9(d), 9(e), and 9(f).

7. CONCLUSIONS

In this paper, we revisited the problem of sequential pattern hiding and proposed a new definition that takes into account both the side-effects and the distortion incurred by sanitization. We also developed two heuristic algorithms for sequence hiding, which are guided by graph-based strategies. Extensive experiments verified that our algorithms incur significantly fewer side-effects and less data distortion than the existing method, while being more efficient. We believe that the graph-based representation can be used to solve other privacy problems related to sequential and trajectory data.

8. REFERENCES