A Two-Phase Algorithm for Mining Sequential Patterns with Differential Privacy

Luca Bonomi
Department of Math&CS
Emory University
Atlanta, USA
lbonomi@mathcs.emory.edu

Li Xiong
Department of Math&CS
Emory University
Atlanta, USA
lxiong@mathcs.emory.edu

ABSTRACT

Frequent sequential pattern mining is a central task in many fields such as biology and finance. However, release of these patterns is raising increasing concerns on individual privacy. In this paper, we study the sequential pattern mining problem under the *differential privacy* framework which provides formal and provable guarantees of privacy. Due to the nature of the differential privacy mechanism which perturbs the frequency results with noise, and the high dimensionality of the pattern space, this mining problem is particularly challenging. In this work, we propose a novel two-phase algorithm for mining both prefixes and substring patterns. In the first phase, our approach takes advantage of the statistical properties of the data to construct a model-based prefix tree which is used to mine prefixes and a candidate set of substring patterns. The frequency of the substring patterns is further refined in the successive phase where we employ a novel transformation of the original data to reduce the perturbation noise. Extensive experiment results using real datasets showed that our approach is effective for mining both substring and prefix patterns in comparison to the state-of-the-art solutions.

Categories and Subject Descriptors

H.2.7 [**Database Administration**]: [Security, integrity, and protection]; H.2.8 [**Database Applications**]: [Data mining]

General Terms

Data mining, Privacy, Performance

Keywords

Differential Privacy, Sequential Data, Frequent Patterns

1. INTRODUCTION

Sequential patterns and sequential data, such as trajectories or DNA sequences, are extensively used in many applications. For example, traffic data collected from major roads could be used to identify the most traveled areas and to predict traffic congestions. In this setting where it is important to capture the sequentiality of the real events in the data, we study the mining of two types of sequential patterns: *substring* and *prefix* patterns. The process of mining substring patterns is extensively used in several applications such as biological sequence analysis [21], and in finding frequent events in sequence of episodes [15]. On the other hand, the mining of prefixes plays a central role in many database tasks, for example in the prefix-filtering [3, 8] where the prefix is used as a criterion for pruning the pair of objects to compute together, and hence to improve the performance.

The rapid increase of data available is not only making the use of sequential patterns appealing in different settings but it is also creating more opportunities to collect personal data which is raising the question of how to protect the individual privacy. In fact, a malicious adversary could make use of sequential patterns to disclose sensitive information of users participating in the data. Recently, Dwork et al. [12] proposed the notion of *differential privacy*, which is based on the idea that adding perturbation noise to the output of a function reveals to an adversary only a limited amount of information about the presence or absence of any individual record. This privacy framework provides strong and provable guarantees of privacy and it has become the de facto standard for research in data privacy. Only recently, several techniques [4, 14, 24] have been proposed for mining frequent patterns under this privacy model. Although these solutions have been shown to be effective in some scenarios, the pattern model used is in the form of itemset which makes these approaches unsuitable for capturing the sequentiality of the events in the data. A first approach for differentially private mining of sequential patterns has been proposed in [10]. This solution partitions the input dataset by exploiting the string prefixes. Due to its nature, this approach is quite effective for prefix patterns but the results for substring patterns are quite poor. Recently, Chen et al. [9] proposed an alternative way for mining sequential patterns. They first reduce the dimensionality of the pattern space by restricting the mining on short patterns (n-grams), and second use the Markov assumption to construct a sanitized dataset using the noisy patterns. This approach works well for mining frequent patterns in the form of substrings, however its utility for prefixes is poor. The main reason is that the n-gram model used to construct the sanitized dataset does not consider the position of the patterns nor makes distinction between substring and prefix patterns. This results in the impossibility to recover the prefixes in the released data.

In this paper, we propose a novel approach to effectively mine the top-k substring patterns without losing information about the frequent prefixes. We observe that mining the substring patterns directly from the data incurs a large perturbation noise due to the presence of long strings. Despite this negative result, in real dataset

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.
Copyright 2013 ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505553.

the majority of the strings are short and only few of them are very long. Therefore, it is reasonable to think that a considerable number of occurrences of the frequent patterns are captured by the short strings. This observation motivates us to first mine a candidate set of patterns from the short strings in the data using a prefix tree approach, and then to refine this set by querying a transformed version of the dataset so that the final impact of the perturbation noise is reduced.

Major Contributions.

- We develop a two-phase algorithm that first generates frequent prefixes and a candidate set of substring patterns and second refines the count of the potential frequent substring patterns. In the first phase, we construct a differentially private prefix tree which is used to generate a candidate set of patterns. This phase reduces the computational complexity of looking for all possible frequent patterns. Successively, we refine the count of the candidate patterns on a transformed dataset to reduce the effect of the perturbation noise on the final results.
- In our first phase, we develop a series of enhancement techniques to construct a differentially private prefix tree. First, we make use of the statistical information from a sample of the data to construct a Markov model which we employ to reduce the amount of perturbation noise in the tree construction. Second, we show that even when a sample of the data is not available, we can take advantage of the released noisy data in the tree construction. Furthermore, we formally study the utility of these approaches providing error bounds for mining prefixes and substrings patterns. In particular, we show that there exists a distinction in the utility of mining these two patterns.
- In our second phase, we propose a novel transformation technique that uses a candidate set of patterns to construct a sketch of the original dataset. We show that in this new representation, we can limit the effect of the perturbation noise due to the privacy mechanism, while preserving the occurrences of the patterns.
- An extensive set of experiments show that our approach provides comparable utility for mining frequent substring patterns with the state-of-the-art in [9], while simultaneously achieving significantly better results for frequent prefixes.

The rest of the paper is organized as follows. In the next section, we describe the related works. Section 3 illustrates the privacy model, and the problem definition. Section 4 presents our two-phase algorithm, and the technical steps in the procedure. Section 5 provides an analysis of the computational complexity and the privacy of our approach. A set of experiments and a direct comparison with the state-of-the-art is reported in Section 6. We finally conclude our paper in Section 7.

2. RELATED WORKS

In this paper, we study the mining of sequential patterns, which is a special case of the pattern model present in the literature. In the data mining community, the pattern mining problem is often associated with patterns in the form of itemsets. Despite the wide range of algorithmic solutions to this problem, only a few approaches [4, 14, 24] study the mining of frequent patterns with differential privacy. However, due to the nature of the patterns, these approaches are not suitable in our setting.

For sequential data, the problem of mining frequent patterns is commonly associated with mining trajectory data. Recently a variety of solutions [1, 2, 5, 19, 23] have been proposed for publishing privacy preserving trajectory data. Abul et al. [1] proposed the notion of (k, δ)-anonymity, which is based on the idea that given a location imprecision δ, the moving objects are indistinguishable if at least k of them are coexisting in the same cylinder of radius δ. In the work in [2], the authors proposed a technique to achieve anonymity in spatiotemporal datasets using spatial generalization and k-anonymity. Another extension of the k-anonymity for moving objects have been proposed by Yarovoy et al. [23]. Terrovitis and Mamoulis [19] proposed a suppression technique whenever a privacy leak occurs, where the leakage is defined as the ability of an adversary of inferring the presence of a location using a set of spatial projections. Although all these techniques have been shown to be effective in several scenarios, their privacy models are not able to provide formal guarantees of privacy.

Under differential privacy, only two techniques have been proposed [9, 10] to tackle the problem of sequential pattern mining. The first approach proposed in [10] uses a differentially private prefix tree to partition the string data and to release a sanitized dataset from which frequent sequential patterns can be mined. Recently Chen et al. [9] proposed a technique based on variable length n-grams and a Markov model to publish a sanitized dataset. This method initially truncates the dataset by keeping only the first l_{max} symbols of the strings in input. Successively this truncated dataset is used to compute the frequency of the variable length patterns (n-grams) up to a fixed length n. Then the information about these patterns is used in a Markov model to release a sanitized dataset. As we discussed earlier, these two approaches work for either prefix mining or substring mining but cannot preserve both patterns.

3. PRELIMINARIES AND PROBLEM DEFINITION

3.1 Differential Privacy

Differential privacy [12] is a recent notion of privacy that aims to protect the disclosure of information when statistical data are released. The differential privacy mechanism guarantees that the computation returned by a randomized algorithm is insensitive to the change in any particular individual record in the input data.

DEFINITION 1 (DIFFERENTIAL PRIVACY [12]). *A non interactive privacy mechanism M gives ϵ-differential privacy if for any two input sets (databases) D_A and D_B with symmetric difference of one (neighboring databases), and for any set of outcomes $S \subseteq Range(M)$,*

$$Pr[M(D_A) \in S] \leq e^{\epsilon} \times Pr[M(D_B) \in S] \qquad (1)$$

The parameter ϵ is the *privacy parameter* (also known as privacy budget) which defines the privacy level of the mechanism. Higher values of ϵ lead to lower level of privacy, while smaller values pose a stronger privacy guarantee. To achieve differential privacy one well established technique is the *Laplace Mechanism* [13]. This strategy is based on the concept of *global sensitivity* [13] of the function to compute.

DEFINITION 2 (GLOBAL SENSITIVITY [13]). *For any two neighboring databases D_A and D_B, the global sensitivity for any function $F : D \rightarrow \mathbb{R}^n$ is defined as:*

$$GS(F) := \max_{D_A, D_B} \|F(D_A) - F(D_B)\|_1 \qquad (2)$$

The Laplace mechanism is used in our paper to construct differentially private algorithms, so we briefly discuss it below. Let F be a function, and ϵ be the privacy parameter, then by adding noise to the result $F(D)$ we obtain a differential privacy mechanism. The noise is generated from a Laplace distribution with probability density function $pdf(x|\lambda) = \frac{1}{2\lambda}e^{-|x|/\lambda}$, where the parameter λ is determined by ϵ and $GS(F)$.

THEOREM 1 (LAPLACE MECHANISM [13]). *For any function $F : D \to \mathbb{R}^n$, the mechanism $M(D)$ that returns:*

$$M(D) = F(D) + Lap(GS(F)/\epsilon) \tag{3}$$

guarantees ϵ-differential privacy.

Two composition properties are extensively used when multiple differential privacy computations are combined. These two properties are known as *sequential* and *parallel* compositions [16].

THEOREM 2 (SEQUENTIAL COMPOSITION [16]). *Let M_i be a non-interactive privacy mechanism which provides ϵ_i-differential privacy. Then, a sequence of $M_i(D)$ over the database D provides $(\sum_i \epsilon_i)$-differential privacy.*

THEOREM 3 (PARALLEL COMPOSITION [16]). *Let M_i be a non-interactive privacy mechanism which provides ϵ_i-differential privacy. Then, a sequence of $M_i(D)$ over disjoint subsets of database D provides $(\max_i \epsilon_i)$-differential privacy.*

In our approach, the noise may not come from a single Laplace distribution, but rather is composed by a sum of independent Laplace distributions. Therefore, here we state two useful results for sum of independent Laplace distributions.

LEMMA 1 (SUM OF LAPLACE DISTRIBUTIONS [7]). *Let $Y = \sum_{i=1}^n l_i$ be the sum of $l_1, \dots, l_n$ independent Laplace random variables with zero mean and parameter b_i for $i = 1, \dots, n$, and $b_{max} = \max\{b_i\}$. Let $\nu \geq \sqrt{\sum_{i=1}^n b_i^2}$, and $0 < \lambda < \frac{2\nu^2}{b_{max}}$. Then $Pr[Y > \lambda] \leq exp\{-\frac{\lambda^2}{8\nu^2}\}$*

COROLLARY 1 (MEASURE CONCENTRATION [7]). *Let Y, $\{b_i\}_i$, λ and b_{max} defined as in Lemma 1. Suppose $0 < \delta < 1$ and $\nu > \max\{\sqrt{\sum_i b_i^2}, b_{max}\sqrt{2 \ln \frac{2}{\delta}}\}$. Then $Pr[|Y| > \nu\sqrt{8 \ln \frac{2}{\delta}}] \leq \delta$*

3.2 Markov Model

An m-order Markov model is a statistical model that can be used to predict the frequency of strings. In particular, an m-order Markov model is based on the *Markov independence assumption* which states that the presence of a particular symbol in a sequence depends only on the previous m symbols. Formally, given a sequence of n observed symbols $X = X_1 X_2 \dots X_n$, and let $m < n$, using the Markov assumption we have that:

$$Pr[X_i = a|X] \approx Pr[X_i = a|X[i-m, i-1] = y[1, m]] \tag{4}$$

A stationary Markov model of order m is completely defined by its transition matrix $\Pi = (\pi(y[1, m], a))$ for $y[1, m] \in \Sigma^m$, where each π is a transition probability defined as follows:

$$\pi(y[1, m], a) = Pr[X_i = a|X[i-m, i-1] = y[1, m]], m < i < n \tag{5}$$

Since the true model is generally unknown the transition probability can be estimated by using the maximum likelihood estimator as in [18], where $f_D(x)$ denotes the frequency of a pattern x in D.

$$\hat{\pi}(y[1, m], a) \approx \frac{f_D(y[1, m]a)}{f_D(y[1, m])} \tag{6}$$

3.3 Problem Definition

In this work, we are interested in mining *sequential patterns* that are in the form of a sequence. A pattern p of length n is represented as a sequence of symbols $p = a_0 a_1 \cdots a_{n-1}$ where each symbol a_i belongs to a finite alphabet Σ. We denote the length of p with $|p|$. We also refer to these sequential patterns as *strings*. Furthermore, we say that a pattern p *occurs* at position i in a string x if there exists $j \in [0, |x| - |p|]$ such that $x_{j+i} = a_i$ for $i = 0, \dots, |p| - 1$. In other words, the substring $x[j, j + |p| - 1]$ matches the pattern p. We introduce the concept of frequency of a pattern as follows.

DEFINITION 3 (FREQUENCY). *For any pattern p we denote by $f_x(p)$ the number of occurrences of p in x. When a set of N strings $D = \{x^0, x^1, \dots, x^{N-1}\}$ is given as input, we define by $f_p := f_D(p) = \sum_{i=0}^{N-1} f_{x^i}(p)$ the frequency of the pattern p in D.*

The definition of frequency based on the number of occurrences of a pattern allows us to capture those repetitive patterns that appear multiple times within the same string. This is crucial for a variety of domains such as time-series data, where the use of the frequency of the patterns within the same series is used to predict the near future values. Our mining problem is formalized below.

PROBLEM 1 (TOP-k MINING PROBLEM). *Given a positive integer k, and a range of pattern lengths I, report the list $\mathcal{F}_I$ of the top-k most frequent patterns of variable length on the input set D.*

In this paper, we consider both *substring* and *prefix* patterns. A substring pattern for a dataset D is represented as any sequence of symbols that is occurring in D. On the other hand, a prefix pattern p is a substring pattern such that there exist some strings in D starting with p. Furthermore, the cardinality of this set represents the occurrence of p in D.

EXAMPLE 1. *Let $D = \{ababbaa, abab, babba\}$ be an input dataset. The pattern $p_1 = aba$ is a prefix pattern for D since both the first and the second string start with p_1. The pattern $p_2 = bba$ is only a substring pattern for D, since no input records start with it. Both these patterns occur twice in D.*

In the rest of the paper, the term pattern will refer to substring pattern if not specified otherwise.

3.3.1 Problem Challenges

In the differentially private mining of frequent sequential patterns there are two major challenges. First, the mining of these patterns is computationally intense. Indeed, the possible number of patterns grows exponentially with the length of the patterns, which makes the mining process inefficient if done naively. Second, the count of occurrences of patterns has high sensitivity, which means that a large amount of perturbation noise is needed to guarantee differential privacy. A formal analysis for the sensitivity of counting occurrences is reported below.

LEMMA 2 (SENSITIVITY FOR COUNTING OCCURRENCES). *Let D be an input dataset with maximum string length l_{max}, then for a query $q = [p_1, p_2, \dots, p_n]$ which for each pattern p_i of length in the range $I = [q_{min}, q_{max}]$ with $1 \leq q_{min} \leq q_{max}$ computes the number of occurrences in D, the sensitivity $GS(q)$ of q is at most $\Delta I \cdot l_{max}$, where $\Delta I = q_{max} - q_{min} + 1$.*

PROOF. *For a fixed pattern length $j \in I$ and a string x, there are at most $|x| - j + 1$ occurrences of patterns of length j in x (either the same repeated or different patterns). Therefore, given the range I, there are at most $\sum_{j=q_{min}}^{q_{max}} (|x| - j + 1) \leq (q_{max} - q_{min} + 1)|x|$ patterns with length in I that can appear in x. Hence, it follows that for any neighboring dataset D' obtained by removing or adding a single string from D the answer for q can change at most by $\Delta I \cdot l_{max}$.* □

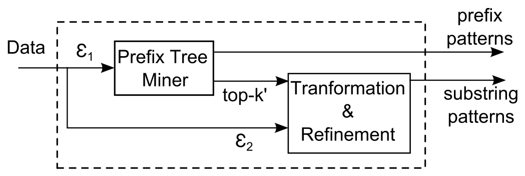


Figure 1: Overview of two-phase algorithm.

Lemma 2 states that the sensitivity of counting occurrences is as high as the length of the longest string in the dataset. Therefore, even if only one very long string is present in the input dataset, all the frequency of the patterns have to be perturbed by a large amount of noise which can drastically reduce the utility.

4. TWO-PHASE ALGORITHM

In this section, we present our two-phase algorithm to address the afore-mentioned challenges. An overview of our approach is illustrated in Figure 1. In the first phase, we use an enhanced **prefix tree miner** to release an ϵ_1-differentially private prefix tree. We will use this tree to first directly mine prefix patterns, and to retrieve a set of top-k' patterns to use in the second phase. In the second phase, we use a **transformation and refinement** step to first construct a sketch of the dataset and second refine the count of the candidate patterns. We use the remaining ϵ_2 privacy budget to query the transformed dataset and retrieve the final counts.

Our strategy presents several advantages. First, the use of a prefix tree allows us to achieve high utility for mining frequent prefixes. Second, the use of a candidate set of patterns help us to limit the number of possible candidate frequent patterns to mine in the second phase. In fact, rather than considering the entire universe of patterns, we focus only on k' candidates. Third, we refine the count of the mined patterns in the top-k' by issuing counting queries on a transformed version of the dataset. In this way, we show that we can control the sensitivity of the counting query leading to a smaller amount of perturbation noise in reporting the final top-k patterns. In the rest of the section, we describe these two phases in details.

4.1 Model-based Prefix Tree Miner

Lemma 2 points out that the sensitivity for counting the occurrences of patterns is related to the maximum length of the strings in the input dataset. This fact can result in loss of accuracy for the mined patterns since even the presence of one very long string in the input forces a large amount of noise. However, we can observe that in real world applications the majority of the strings in the data are short and only few of them are very long. Consider for example the Anonymous Web Data `MSNBC`, where each sequence in the dataset corresponds to page views of a user during that twenty-four hour period, the average length for the strings is only 4.7 symbols while the maximum length is 14975. Intuitively, since most of the strings are short it is reasonable to assume that the occurrences for the frequent patterns are probably mostly captured by the short strings rather than the long ones. This observation motivates us to consider an alternative way to mine the patterns. In particular, we want to privilege the short strings over the long ones by processing the strings in the datasets starting from their prefix. A baseline approach has been firstly proposed for mining trajectory data in [10, 11] and further extended and applied in a different setting in [6]. This approach starts from a tree formed by a root node representing the entire dataset, and for each level in the tree a new node ν is added according to the prefix of the strings in the data. Given a maximum height h_{max} for the tree, a node representing a prefix ω is extended with the symbols in the alphabet Σ creating a new node ν. For each node ν, a privacy parameter ϵ_ν is assigned and the noisy count of strings in the partition associated with ν is com-

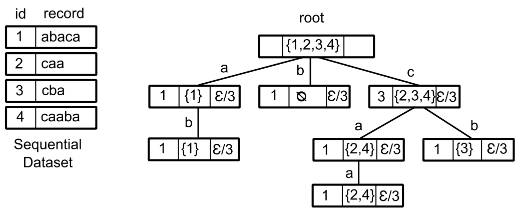


Figure 2: Example of prefix tree, where each label represents a symbol in the prefix. Each node has a noisy count, a partition of strings sharing the same prefix, and a privacy budget.

Algorithm 1 Private Prefix-Tree Sample Miner

```
1:  procedure PT-SAMPLE($D$, $\epsilon_1$, $S$)
    Input: dataset $D$; privacy parameter $\epsilon_1$; Sample $S$
    Output: $\mathcal{T}$ private Prefix-Tree

2:      Construct a Markov Model of order $m$ from the sample $S$
3:      $\mathcal{T}$ formed by the $root$
4:      Create a queue $Q$
5:      $Q \leftarrow root$
6:      while ($Q$ is not empty) do
7:          $v \leftarrow Q.remove$
8:          for (every symbol $a$ in the alphabet $\Sigma$) do
9:              Extend the prefix $\omega$ of the node $v$ with the symbol $a$
10:             Create a possible child $\nu$ of $v$
11:             $\tilde{c}(\omega a) \approx \tilde{c}(\omega)\frac{f_S(\omega[1,m]a)}{f_S(\omega[1,m])}$        ▷ Model Estimate
12:             $\epsilon_\nu \leftarrow$ BUDGET ALLOCATION($\nu$, $\lambda$, $\epsilon_{acc}$, $\epsilon_1$)
13:             Calculate the noisy count for $\nu$
14:             Decide to extend $\nu$ or mark it as a leaf node
15:             Add $\nu$ to $Q$
16:         end for
17:     end while
18:     Enforce consistency constraint on $\mathcal{T}$
19: end procedure
```

puted by perturbing the real count $c(\nu)$ with Laplace noise with parameter ϵ_ν (i.e. the sensitivity of counting prefixes is 1). To reduce the computational cost only partitions of size greater than a threshold value θ are further refined. An illustration of this prefix tree approach with total privacy parameter ϵ is reported in Figure 2. We denote this approach as PT-Base.

Inspired by this technique, we introduce an enhanced prefix tree mining algorithm which uses statistical properties of the data to maximize the quality of the counts in the nodes of the tree. Our goal consists in using the information from a statistical model to calibrate the noise injected with the Laplace Mechanism so that we minimize the effect of the perturbation noise on the overall utility. First we develop a solution based on a sample of the real data, denoted as PT-Sample. Second, we propose an alternative solution to handle the case where this a priori information is not available, we call this algorithm PT-Dynamic.

4.1.1 A Sample Based Model

In the construction of the prefix tree we employ the Laplace Mechanism combined with the estimated count from a Markov model constructed on a sample input. In many scenarios, we can assume that together with the sensitive data we have some publicly available information that we can use in our mining process. Consider for example the case of medical data, many patients prefer to keep their privacy right on their data but it often occurs that a part of them opt to share their data with consent. Such information can be captured by a statistical model and used in the construction process of the prefix tree.

One of the challenges in using the Laplace Mechanism consists of the fact that the noise injected to achieve ϵ_1-differential privacy depends only on the value of ϵ_1 and sensitivity of the query. Since the magnitude of the noise does not depend on the real answer, the mechanism turns out to be in favor of queries with a large count. Indeed, given a fixed amount of privacy budget ϵ_1 we will incur a

Algorithm 2 Budget Allocation

```
1: procedure BUDGET ALLOCATION($\nu$, $\lambda$, $\epsilon_{acc}$, $\epsilon_1$)
   Input: current node $\nu$; threshold $\lambda$; privacy budget used till the parent of $\nu$ $\epsilon_{acc}$;
   privacy parameter $\epsilon_1$
   Output: $\epsilon_\nu$ privacy for the node $\nu$

2:     $\tilde{c}(\nu) \leftarrow \nu.model\_count$
3:     $\bar{\epsilon} \leftarrow (\epsilon_1 - \epsilon_{acc})/(h_{max} - \nu.h)$
4:     if ($\tilde{c}(\nu) < \lambda$) then
5:         return $\bar{\epsilon}$
6:     else
7:         return $\bar{\epsilon}\frac{\lambda}{\tilde{c}(\nu)}$
8:     end if
9: end procedure
```

larger error for those queries with smaller values of count. Therefore, if in our tree we want to guarantee overall ϵ_1-differential privacy, the use of an equal amount of privacy budget among the nodes will penalize those with smaller counts. To mitigate this phenomena, we first construct a Markov model from a sample, and we use the model to obtain an estimated count for each node. Second, we use this information to properly decide the amount of noise to inject so that the overall prefix tree satisfies ϵ_1-differential privacy and the error introduced by the noise is reduced. For example, in any root-to-leaf path instead of using a fixed amount of privacy budget among the path, we can spend a smaller budget for nodes with large count and use this saving later in the path for nodes with smaller counts. Our procedure is illustrated in Algorithm 1.

Budget Allocation. We start assigning a privacy budget $\bar{\epsilon}$ for nodes associated to prefixes of length less than m (the length of the model) which assumes the default value of ϵ_1/h_{max}. This initialization step is motivated by the fact that for these nodes the model cannot provide a good estimated count, so it is better to use the Laplace Mechanism directly with parameter $\bar{\epsilon}$. Then in our top-down partitioning process, for a node ν with prefix ωa of length $l+1$, we first compute an estimated count using the model as in line 11 of Algorithm 1. Then when the estimated count $\tilde{c}(\nu) = \tilde{c}(\omega a)$ is returned, we compare it against a threshold value $\lambda = \alpha(|\tilde{D}|/|P_l|)$, where α is a constant, $|\tilde{D}|$ is the noisy size of the dataset, and P_l is the set of prefixes of length l in the tree. The value of λ gives an indication about the average counts of the prefixes in the previous level in the tree, and it is used to decide the amount of privacy budget for the node ν. The privacy budget allocation strategy is illustrated in Algorithm 2.

The procedure computes for a node ν the amount of privacy budget to allocate. Given the amount of privacy budget allocated so far on the path from the root to the parent node of ν, the algorithm starts with a privacy budget $\bar{\epsilon}$ as in line 3. This value is defined as the privacy budget left on the path divided by the remaining level in the tree. Then if the estimated count $\tilde{c}(\nu)$ is smaller than the threshold λ this privacy parameter is directly returned, otherwise it will be scaled by a factor $\lambda/\tilde{c}(\nu)$, which is in the range $(0, 1)$. In this way, we can spend a smaller amount of budget for the frequent prefixes which leads to a better use of the overall privacy parameter, hence higher utility. Note that if already the overall privacy budget has been consumed (i.e. $\epsilon_{acc} = \epsilon_1$), then no further privacy parameter ϵ_ν will be allocated.

Noisy Count Computation. After the privacy parameter ϵ_ν is determinate, the noisy count for the node ν is computed by perturbing the real count of ν with Laplace noise with parameter ϵ_ν. This perturbed count will be assigned as the final count for the prefix represented by the node ν if the count from the model is smaller than λ. Otherwise, we use as a final count the average between the noisy count and the count from the model. Notice that, if we could have an estimate of the error in our Markov model at this step, we could employ a weighted average to determine the count with minimum variance. However, without this information we decide to use the simple average which from our experiments provides fair results.

Finally, to decide if a node is a leaf or an internal node in the tree, we compare its noisy count against the threshold value θ. If the current node is a leaf, we use up all remaining privacy budget to refine its count.

4.1.2 A Dynamic Model

In the previous section, we presented a construction process which uses the statistical properties from a sample of the data to calibrate the noise in the tree. However, in certain settings a sample of the data or background knowledge are not always available. In such cases we cannot rely on a priori information. In this section we show how to modify our previous technique so that during the tree construction process we can dynamically use the partial tree to construct a statical model. We call this algorithm PT-Dynamic. The idea of using some information about noisy data to improve the utility has been shown to be effective in reducing the relative error for count queries in the recent work of Xiao et al. [22].

Our strategy follows similar steps as in Algorithm 1, where now every time a new node is attached to the prefix tree the statistical information are updated. Therefore, now the Markov model is defined on the noisy frequencies generated during the construction of the prefix tree. In this case the estimated frequency of a prefix ωa is computed using the transition probability obtained from the noisy frequencies released in the previous levels of the tree as follows:

$$\tilde{c}(\omega a) \approx \tilde{c}(\omega)\frac{\hat{f}_D(y[1,m]a)}{\hat{f}_D(y[1,m])} \qquad (7)$$

We can observe that for each new node added to the prefix tree, the only information that has to be updated are the frequencies $\hat{f}_D(y[1,m]a)$ and $\hat{f}_D(y[1,m])$ which can be efficiently computed.

4.1.3 Error Analysis

In this section, we investigate the error in the counts reported by the prefix tree for two types of patterns: *prefixes* and *substrings*. In particular, we would like our mining algorithms to be useful, that is, their output should well approximate the real count of the pattern (prefix and substring) on the input data. Below, we formally define the notion of utility for counting query.

DEFINITION 4 ((ξ, δ)-USEFUL). *A mining/counting algorithm $\mathcal{A}$ is (ξ, δ)-useful, if for any data input D and pattern p, with probability at least $1 - \delta$, the approximate answer from $\mathcal{A}$ is within ξ from the real count of p.*

Prefix Patterns. The counts for the prefixes can be directly retrieved from the node in the prefix tree. Any prefix count query for a pattern p, can be answered using the prefix tree by returning the noisy count of the node with prefix label p. We quantify the error bound of the noisy frequency in the following Corollary.

COROLLARY 2 (ERROR BOUND FOR PREFIX QUERY). *For any prefix count query for a pattern p, the noisy count $\tilde{c}(p)$ associated to the node in the tree having label p and privacy parameter ϵ_i, with probability at least $1 - \delta$, the quantity $\xi = \|c(p) - \tilde{c}(p)\|_1$ is at most $O(\frac{1}{\epsilon_i} log \frac{1}{\delta})$.*

PROOF. *It follows from the tree construction, and the pdf of Laplace distribution.* □

Substring Patterns. In our tree representation, the mining of substring patterns is a more challenging problem than mining prefixes.

While there is a one to one correspondence between the frequent prefix pattern and the node in the tree, for frequent substring patterns this relationship is more complex. Indeed, the frequency of a pattern p is computed as the sum of the noisy count of the prefixes where p occurs as a suffix. Below we quantify the noise accumulated in this process, which will help us to understand the theoretical limitations of this approach.

Formally, let $\tilde{f}_p$ be the frequency released by the prefix tree for the pattern p, and denote by f_p its real frequency. Let n denote the number of nodes in the tree having p as a suffix, then we can write $\tilde{f}_p$ as follows.

$$\tilde{f}_p = \sum_{i=0}^{n-1} \tilde{c}(\nu_i) = \sum_{i=0}^{n-1} c(\nu_i) + \sum_{i=0}^{n-1} Lap(1/\epsilon_i) \tag{8}$$

In the construction of the tree, some prefixes are not extended due to the fact that their noisy counts do not pass the threshold value θ. Therefore, the sum of the counts for the prefixes in the tree containing p as a suffix can be upper bounded by the frequency of p in the real data f_p as follows.

$$\tilde{f}_p \leq f_p + \sum_{i=0}^{n-1} Lap(1/\epsilon_i) \tag{9}$$

We denote the sum of these Laplace noises with the random variable $Y = \sum_{i=0}^{n-1} Lap(1/\epsilon_i)$. Therefore, the variable Y determines the error in estimating the absolute value of the frequency $\xi = \|\tilde{f}_p - f_p\|_1$. Below, we characterized this quantity.

COROLLARY 3 (ERROR BOUND FOR SUBSTRING QUERY). *For any substring count query for the pattern p, the noisy count $\tilde{f}_p$ obtained by summing the noisy count of the n nodes having p as a suffix, with probability at least $1 - \delta$, the quantity $\xi = \|f_p - \tilde{f}_p\|_1$ is at most $O(\sqrt{\sum_{i=0}^{n-1} \frac{1}{\epsilon_i^2}} log\frac{1}{\delta})$.*

PROOF. *The proof follows from Corollary 1, where we choose $\nu = \sqrt{\sum_{i=0}^{n-1} b_i^2}\sqrt{2\ln\frac{2}{\delta}}$.* $\square$

This result shows a distinction between the utility for prefix and substring patterns. As we expected the error for mining prefixes is small, while for substring we accumulate noises from multiple nodes. This motivate us to use the prefix tree to mine the prefixes directly and use a second phase to improve the final results for the substring patterns.

4.2 Transformation & Refinement

In the previous analysis we pointed out that counting the occurrences of substring patterns from the tree could incur a large error due to the sum of multiple perturbation noises. This could result in poor performance if we just mine the top-k in this phase. However, due to the nature of the distribution of the frequency for the frequent patterns, we can use the prefix tree to generate a candidate set of k' patterns with $k' > k$, which likely contains the real top-k patterns. Therefore, the goal of our second phase consists in finding the top-k patterns from the set of candidates.

In this phase, we refine the count of the patterns issuing a counting query, however due to its high sensitivity we could incur a large perturbation noise if it is applied on the original dataset directly, as shown in Lemma 2. Therefore, our idea consists in introducing a new representation of the original dataset where we can control the sensitivity of the count query and at the same time preserve the frequent patterns. This transformation process is based on the concept of *fingerprint* which is extensively applied in string matchings [20, 17]. In our paper, we define fingerprint as follows.

DEFINITION 5 (FINGERPRINT). *Given a set of patterns $\mathcal{C} = \{p_1, p_2, \ldots, p_n\}$, for a string x we call the vector $\mathbf{fp}_{\mathcal{C}}(x)$ the fingerprint of x on $\mathcal{C}$, where the i-th component ($\mathbf{fp}_{\mathcal{C}}(x)[i]$) represents the number of occurrences of p_i in x.*

Intuitively, the fingerprint of a string represents the contribution of the string on the occurrences of the patterns in $\mathcal{C}$. Our idea is to represent the original strings using their fingerprints on the top-k' patterns. In particular, given a truncated maximum length l'_{max}, for strings of length smaller than l'_{max} we keep their fingerprint directly, otherwise we construct a fingerprint that is as close as possible to the original one and can be represented with a string of length l'_{max}. In this way, we bound the sensitivity of the count query for the patterns on the transformed dataset to be dependent on l'_{max}, which can be considerably smaller than the maximum length in the original strings. Clearly, this process may lead to approximation error in representing long strings as illustrated in the Example below.

EXAMPLE 2. *Let $\mathcal{C} = \{aa, ab, bb, ac\}$ be a set of patterns, and let $x = aabb$ and $y = bbcacdcbccddaa$ be two strings. Given a maximum length $l'_{max} = 4$ we want to represent these stings with the vectors $\bar{x}$ and $\bar{y}$ respectively. First, for the string x we can notice that the constraint on the maximum length is satisfied therefore we use $\bar{x} = \mathbf{fp}_{\mathcal{C}}(x) = [1, 1, 1, 0]$. Second, for the string y the length constraint is violated, so we represent y with a vector $\bar{y} = [0, 0, 1, 1]$ which is a fingerprint of a string of length l'_{max} and has minimum distance from $\mathbf{fp}_{\mathcal{C}}(y) = [1, 0, 1, 1]$. Overall, with this representation we lose one occurrence of the pattern aa in the string y.*

As Example 2 pointed out, the transformation process may introduce an approximation error in representing long strings since some of the occurrences of the patterns are lost. Therefore, it is important to reduce this error. Here we formalize the fingerprint construction as the following optimization problem.

PROBLEM 2 (OPTIMAL CONSTRAINED FINGERPRINT). *Given a maximum length l'_{max}, a set of patterns $\mathcal{C}$, and an input string x with fingerprint $\mathbf{fp}_{\mathcal{C}}(x)$, find the constrained fingerprint vector $\bar{x}^* = \mathbf{fp}_{\mathcal{C}}(x^*)$, where x^* is defined below.*

$$x^* = \underset{y \in \Sigma^{l'_{max}}}{\arg\min} \|\mathbf{fp}_{\mathcal{C}}(y) - \mathbf{fp}_{\mathcal{C}}(x)\| \tag{10}$$

In the rest of section, we investigate the challenges for designing an efficient and effective transformation algorithm to reduce the approximation error, and a simple process for determining a suitable value for k' and l'_{max}.

4.2.1 Fingerprint Construction

In this section, we propose a heuristic strategy which decomposes the input string into blocks with an associated profit.

DEFINITION 6 (BLOCK PROFIT). *Given a block b and a set of patterns $\mathcal{C}$, the profit of b is defined as the ratio between the number of occurrences in b of the candidates in $\mathcal{C}$, over the length of b.*

When a string is received in input, we will return a vector by computing the fingerprint of the string formed by selecting the most profitable blocks till a maximum length is reached. Our transformation procedure is illustrated in Algorithm 3. We divide the input string x into $|x| - b_{min} + 1$ consecutively overlapping blocks of length b_{min}, which is defined as the minimum length of the patterns in $\mathcal{C}$. From line 8 to 10 in Algorithm 3, we scan the blocks

Algorithm 3 String Transformation Procedure

```
1: procedure TRANSFORMATION($x$, $\mathcal{C}$, $l'_{max}$)
     Input: string $x$; set of frequent patterns $\mathcal{C}$; maximum length $l'_{max}$
     Output: $\bar{x}$ vector representation

2:     $\bar{x} \leftarrow 0$
3:     if ($|x| \leq l_{max}$) then
4:         $\bar{x} \leftarrow \mathbf{fp}_{\mathcal{C}}(x)$
5:         return $\bar{x}$
6:     end if
7:     Let $\mathcal{B}$ be the set of blocks for $x$ initialized with their profit
8:     for (every block $i$ in $\mathcal{B}$) do
9:         Merge two consecutive non-empty blocks $i$ and $i+1$ into $i$
10:    end for
11:    Update the contribution of the merged blocks
12:    Sort the blocks in decreasing order according to the profit
13:    $\mathcal{S} \leftarrow$ SELECT BLOCKS($\mathcal{B}, l'_{max}$)
14:    for (every block $i$ in $\mathcal{S}$) do
15:        Update $\bar{x}$ with the contribution of $i$ on the pattern in $\mathcal{C}$
16:    end for
17:    return $\bar{x}$
18: end procedure
```

and we merge two consecutive blocks if the merged block contains some occurrences of the patterns in $\mathcal{C}$. Then at line 13, the algorithm selects the most profitable blocks (i.e. those with the most contribution to the occurrences of the patterns in $\mathcal{C}$) till the accumulated length reaches l'_{max}. Note that if the length exceeds this threshold, we truncate part of the block. We finally combine the contributions of the blocks selected and return the fingerprint.

4.2.2 Refinement

In the previous section we developed a transformation strategy which maps the original set of strings D into a set $\bar{D}$ of vectors (fingerprints). In this new representation, the count of the occurrences for a pattern p_i in $\mathcal{C}$ is computed by summing up the i-th component of all the vectors in the transformed data $\bar{D}$. Specifically, given the set of candidate patterns $\mathcal{C}$ and the transformed dataset $\bar{D}$, we will perform the query $q = [p_1, p_2, \ldots, p_{k'}]$ on $\bar{D}$, where $p_i \in \mathcal{C}$, for $i = 1, 2, \ldots, k'$. Since these vectors represent fingerprints of strings whose lengths are bounded by l'_{max}, it follows from Lemma 2 that it is sufficient to perturb the answer of q by adding Laplace noise with parameter $\epsilon_2/(\Delta I \cdot l'_{max})$ to achieve ϵ_2-differential privacy for the query q on $\bar{D}$. We use these noisy counts to identify the top-k patterns from the set $\mathcal{C}$.

In Section 5.2, we will explain the privacy implication of this mechanism with respect to the original dataset D, in particular we show that our transformation and refinement steps guarantee ϵ_2-differential privacy also for the original dataset.

4.2.3 Parameter Selection

In this section, we investigate how to select the parameters used in our two-phase algorithm. We defer the study of the impact of ϵ_1 to the experiments section.

Choosing k'. In our first phase, we use our prefix mining algorithms to mine a set of candidate patterns. It is crucial in this step to choose a suitable value of k' so that with high probability the real top-k patterns are contained in the candidate set. Here, we propose a simple way to select the value for k'. We first make some assumptions on the distribution of the patterns. In particular, we assume that the distribution of the frequency of the patterns follows a Zipf's distribution $z(k; s, m)$, where s is a parameter related to the specific dataset and m denote the number of patterns considered. This assumption is motivated by the fact that in many real scenarios the frequency of the pattern follows a power-law distribution (e.g. frequency of words in English). Furthermore we assume that we can estimate a maximum relative error $\Delta Freq$ for the real frequency of the patterns reported in the first phase. Then, we can find a k' such that the frequency $f_{k'}$ (i.e. frequency of the k'-th pattern) is at most $f_k(1 - \Delta Freq)$. Intuitively, this means that even in the worst case, that is the real frequencies of the top-k patterns decrease by a quantity $\Delta Freq$, the real top-k will be still present in the top-k' patterns. Therefore, we can show that it is sufficient to choose a value of k' that satisfies the following inequality.

$$f_k(1 - \Delta Freq) \geq f_{k'} \tag{11}$$

Then normalizing by the sum of all the frequency we obtain:

$$\frac{f_k}{\sum_{i=1}^{m} f_i}(1 - \Delta Freq) \geq \frac{f_{k'}}{\sum_{i=1}^{m} f_i} \tag{12}$$

Using the assumption of the Zipf's distribution (here we assume $s = 1$ for simplicity), we can rewrite the previous inequality as follows.

$$\frac{1}{k}(1 - \Delta Freq) \geq \frac{1}{k'} \tag{13}$$

Solving the inequality above, we obtain that $k' \geq \frac{k}{1-\Delta Freq} = k(1+\gamma)$, with $\gamma = \frac{\Delta Freq}{1-\Delta Freq}$. Unfortunately, the value of $\Delta Freq$ is not easy to estimate, therefore for simplicity in our approach we set $\gamma = 0.5$.

Choosing l'_{max}. In our two-phase algorithm both the parameters k' and l'_{max} play an important role in the final utility. With larger values of k' it is more likely that the top-k patterns are contained in the candidate set. On the other hand, a smaller value of l'_{max} reduces the amount of noise injected on the final count, but it may introduce a larger approximation error since some occurrences of the candidate patterns may not be captured in the transformation. Although we saw how to compute a value of k', the choice of the optimal value for l'_{max} is still very challenging. Therefore, we determine the maximal length l'_{max} in a heuristic way by setting $l'_{max} = \min\{l^*, L\}$. The length l^* is a value that can be computed from the data, for example in our simulation we choose l^* such that the percentage of the strings with length no greater than l^* is at least 85%. The parameter L instead represents an upper bound on the length which determines a maximum value of the error introduced by the noise in computing the final counts.

5. ANALYSIS

5.1 Complexity Analysis

Prefix Tree phase. The PT-Sample and PT-Dynamic algorithm have running time proportional to the number of nodes in the prefix tree $\mathcal{T}$. First, we can notice that on the level i, we have at most all the possible prefixes of length i defined on the alphabet Σ. Hence, the total number of nodes at level i is bounded by $O(|\Sigma|^i)$. This bound is quite loose, in fact the number of nodes can be much smaller since some prefixes are not extended due to the fact that their counts do not pass the threshold value θ. Moreover, each node performs a counting query on its partition that requires a linear scan. Therefore each level i requires $O(N|\Sigma|^i)$ operations, where N is the size of the database in input. Since, in the tree there are at most h_{max} levels, the overall running time for our algorithms is $O(N|\Sigma|^{h_{max}+1})$. After the tree is constructed, the consistency constraints require $O(h_{max}^2 N)$ operations as shown in [11]. Finally, the running time for traversing the prefix tree is linear with the number of internal nodes in the tree, hence the overall complexity of our algorithms is $O(N|\Sigma|^{h_{max}+1})$.

Transformation and Refinement. The computational complexity for the Algorithm 3 plays an important role on the overall performance since it is applied on each string in the dataset. Therefore, we reduced its running time as follows. First, for each position i in the input string we keep the set of patterns of $\mathcal{C}$ that have an occurrence in i. This can be done in linear time with the length of the string in input when a proper index structure (e.g. prefix tree) is employed for the patterns in $\mathcal{C}$. Second, for each block we store its fingerprint so that the final vector is computed by summing all the fingerprints of the blocks selected. Let $l = |x|$ denote the length of the string x in input, and let $|\mathcal{C}|$ be the size of the set $\mathcal{C}$. Since the number of blocks in the input string is at most $O(l)$, then lines 7 to 11 in Algorithm 3 require linear time, while sorting the blocks requires $O(l \ln l)$. The selection of the blocks at line 13 requires $O(l)$ time since the blocks have to be scanned, while the final construction of the vector at line 14 requires $O(l|\mathcal{C}|)$. Therefore the final complexity for the Algorithm 3 is $O(l(|\mathcal{C}| + \ln l))$. Given N the size of the dataset in input, and l_{max} the maximum length of the strings in input, the complexity for the overall transformation step is $O(Nl_{max}(|\mathcal{C}| + \ln l_{max}))$. The refinement step requires only a linear scan on the transformed data.

5.2 Privacy Analysis

Given the privacy parameters ϵ_1 and ϵ_2, we first show that our two phases achieve ϵ_1 and ϵ_2 differential privacy respectively. Second, we prove that our overall solution satisfies $(\epsilon_1+\epsilon_2)$-differential privacy.

THEOREM 4 (PREFIX TREE MINER ϵ_1-PRIVACY). *PT-Sample and PT-Dynamic algorithm satisfy ϵ_1-differential privacy.*

PROOF. *(Sketch) Our PT-Sample and PT-Dynamic satisfy ϵ_1-differential privacy, due to the fact that the counting queries for constructing the prefix tree are issued on disjoint partitions. So from Theorem 3, it is sufficient to guarantee that on any root-to-leaf path in the tree the maxim privacy budget allocated is at most ϵ_1, this follows directly from Algorithm 2.* □

For the second phase we have the following result.

THEOREM 5 (TRANSFORMATION & REFINEMENT ϵ_2-PRIVACY). *The transformation and refinement steps satisfy ϵ_2-differential privacy.*

PROOF. *(Sketch) We saw in Section 4.2.2 that the refinement step satisfies ϵ_2-differential privacy with respect to the transformed dataset $\bar{D}$. Therefore, what is left to show is that the transformation process preserves the differential privacy. Intuitively, the set of patterns $\mathcal{C}$ used in this step satisfies differential privacy so it does not disclose information about the individual record, however the procedure accesses the original string data directly. It turns out that as long as the transformation is* **local***, that is the output only depends on the individual input string, then applying a ϵ_2-differential privacy mechanism on the transformed dataset $\bar{D}$ also guarantees ϵ_2-differential privacy for the original dataset D. This result has been shown in [24]. Hence our second phase satisfies ϵ_2-differential privacy.* □

Therefore, using the sequential composition property in Theorem 2 on this sequence of privacy mechanisms, it follows that our overall mining approach satisfies $(\epsilon_1 + \epsilon_2)$-differential privacy.

THEOREM 6 (TWO-PHASE ϵ-PRIVACY). *Let ϵ_1 and ϵ_2 be the privacy budgets for the first phase and second phase respectively, with $\epsilon_1+\epsilon_2 \le \epsilon$. Then, the two-phase algorithm satisfies ϵ-privacy.*

Dataset	size	$\|\Sigma\|$	l_{max}	l_{avg}
MSNBC	989,818	17	14975	4.7
house_power	40,691	21	50	50

Table 1: Datasets characteristics

Parameter	Description	value
ϵ	Total privacy parameter	0.1
ϵ_1	Privacy parameter in the first phase	0.85ϵ
I	Range of variable lengths	[2,7]
h_{max}	Depth of the prefix tree	10
k	Number of frequent patterns to mine	60
$\|S\|$	Sample size for PT-S*	10% of $\|D\|$

Table 2: Parameters

6. EXPERIMENTS

We evaluate the performances of our mining algorithms for two types of patterns: prefix and substring. Furthermore, we investigate the impact of specific parameters (privacy level in the first phase ϵ_1, and depth of the prefix tree) on our final results. Finally, we compare our approaches with the n-gram method proposed in [9], which to the best of our knowledge represents the most related work and the state-of-the-art for mining sequential patterns with differential privacy.

Data. In the experiment section we test our approaches on two real sequential datasets. The first is the MSNBC dataset, where each string represents a sequence of web pages visited by users within 24 hours on the msn.com domain. The second is the house_power dataset which is derived from a numeric dataset representing the energy power consumptions of individual household over 47 months. The original data appears as time-series, therefore for our sequential data we first discretized these values and we successively construct a string record from every 50 samples. A summary of the two datasets is reported in Table 1. Both the MSNBC and the household energy consumption datasets are available at the UCI machine learning repository [1].

General Settings. Our two-phase algorithms are denoted by PT-S* and PT-D*, where as the first phase we employ the sample based (Section 4.1.1) and the dynamic model tree (Section 4.1.2) respectively. If not specified in the text the parameters for the algorithms assume the default values reported in Table 2. The utility measure in these experiments is the F_1 score, which is a combination of precision and recall for the mined top-k patterns.

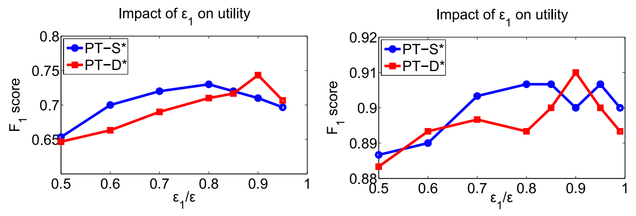
6.1 Impact of the parameters on the utility

First of all, we start to study the impact of our specific parameters on the final utility of the mined results.

Allocation of the privacy budget ϵ_1. The impact of the privacy budget ϵ_1 that we use in the first phase of our algorithms on the final utility is illustrated in Figure 3. We can notice that in both datasets increasing ϵ_1 has beneficial impact on the utility; however, after a certain value an increment in this parameter quickly degenerates the results. This phenomena is caused by the tradeoff between the quality of the candidate set that we generate in the first phase and the noise injected in the second phase. Figure 3 is an indication of the effectiveness of our transformation, since even using a small privacy budget (15%-20% of total ϵ) in the second phase it does not compromise the final results. Therefore in our experiments, we will use $\epsilon_1 = 0.85\epsilon$.

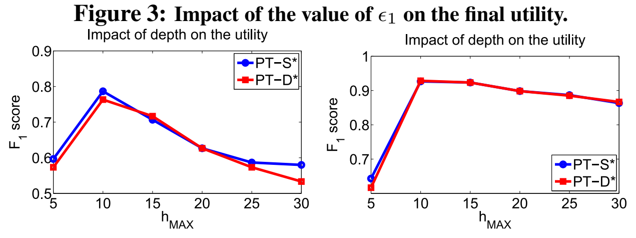
Depth of the tree h_{max}. Figure 4 illustrates the impact of the depth in the prefix tree used in the first phase on the final utility. Ideally, we would like to have a prefix tree as high as possible to capture all the patterns in the data. However, as the height of the tree increases, less privacy budget is available for each node which leads

[1]http://archive.ics.uci.edu/ml/datasets



(a) `house_power` (b) `MSNBC`

Figure 3: Impact of the value of ϵ_1 on the final utility.



(a) `house_power` (b) `MSNBC`

Figure 4: Impact of the depth of the prefix tree on the final utility.

to a larger perturbation noise in the counts. This is more evident in the `house_power` dataset since it is smaller than `MSNBC` (i.e. smaller counts). In our experiments, we fix $h_{max} = 10$ since with this setting our algorithms well perform on both datasets.

6.2 Comparison for mining frequent patterns

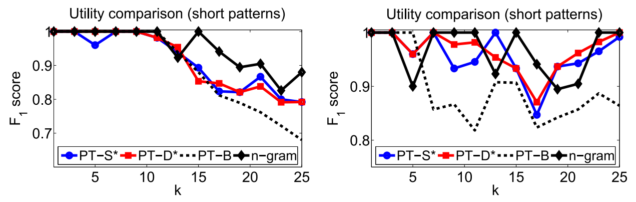
In this part of the experiments, we evaluate our algorithms in mining patterns against the state-of-the art approach n-grams proposed in [9] and with the prefix tree base algorithm PT-B (base line) [10, 11] that we briefly described in Section 4.1. For the n-grams approach we use the default setting suggested by the authors in [9], while for the PT-B we set the maximum depth of the tree equal to the value of h_{max} in our two-phase algorithm. In our experiments, we consider the mining task both for substring and prefix patterns.

6.2.1 Mining Frequent Substrings

To understand the performance of our mining algorithms for substring patterns, we study several scenarios. We first consider the task of mining short frequent patterns with small k values, that is the case where the frequencies of the patterns is more likely to be well separated. In the second case, we study the problem of mining longer patterns with large k values, which is more challenging since even a small perturbation noise in the frequency could drastically change the top-k patterns. We also study the impact of the privacy parameter and the length of the patterns on the final utility.

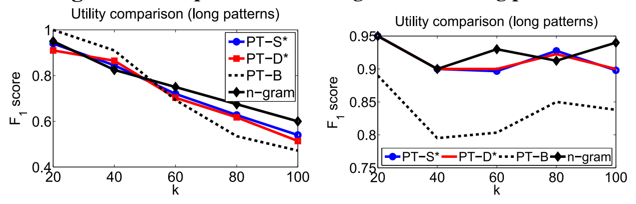
Mining Short Patterns. We consider patterns p of variable length in the range $[2, 3]$ symbols. The results on the two datasets are reported in Figure 5. For the `house_power` dataset the approaches have similar performance for $k < 15$. For k in the range $[15, 25]$ all the approaches decrease their performance. The results for PT-B quickly drop, while our solutions are quite stable. In this range of k values, the n-gram approach performs slightly better than our solutions. For the `MSNBC` the results from our algorithms are comparable with those from the n-gram method. Furthermore, we can see that the base line suffers in both datasets.

Mining Long Patterns. Contrary to the previous setting, we now consider longer patterns. In particular we focus on mining patterns of length in the range $[2, 7]$ for k values between 20 and 100. Figure 6(a) shows the utility on the `house_power` dataset. We observe that our mining techniques constantly provide higher utility than the n-gram strategy for values of k in between 20 and 60, while for larger values this gap becomes smaller. The baseline provides comparable results only for small k, then its performance



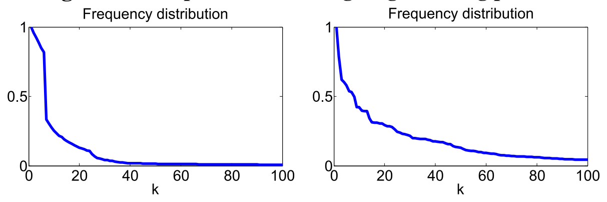
(a) `house_power` (b) `MSNBC`

Figure 5: Comparison for mining short substring patterns.



(a) `house_power` (b) `MSNBC`

Figure 6: Comparison for mining long substring patterns.



(a) `house_power` (b) `MSNBC`

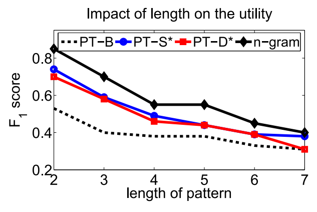
Figure 7: Frequency distribution of long substring patterns.

rapidly decreases. The results for the `MSNBC` dataset are reported in Figure 6(b). In this case our strategies closely follow the results from the n-gram miner, while the baseline constantly suffers providing 10% less utility than our approaches.

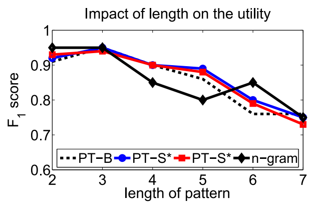
From these mining results we can observe a considerable gap in the utility between the patterns mined from `house_power` and `MSNBC`. In fact, we can see from Figure 7(a) that the frequency distribution of the patterns in the first dataset rapidly decreases while in web browsing data this behavior is not so extreme, as shown in Figure 7(b). This does not provides a good separation between the frequencies of the patterns for large values of k (e.g. $k > 40$) which makes the mining of the patterns in `house_power` more challenging.

Length of the patterns. In the mining process we consider patterns of variable length, so it is important to understand how the length of the patterns affects the final utility result. Intuitively, long patterns are more difficult to mine for two reasons. First, their absolute value of frequency is lower than short patterns. Second, in general there is not a good separation in term of frequency. Hence, the perturbation noise is more likely to drastically change their frequency and shuffle their order in the top-k. We can observe this phenomena from Figure 8, where increasing the length of the patterns mined makes the utility to quickly drop. Our approach well performs against the baseline in both datasets. Compared with the n-gram our solutions provide slightly lower results in the `house_power` dataset while in the `MSNBC` our results are often better.

Total privacy budget ϵ. The impact of the total privacy parameter ϵ on the final utility is reported in Figure 9. As the intuition suggests, we can see that increasing the privacy budget (less privacy) increases the utility. In the `house_power` dataset all the approaches are close, while in the `MSNBC` there is a distinction. In particular, compared with the n-gram technique our solutions provide better utility for small privacy budget, while for $\epsilon > 0.1$ our

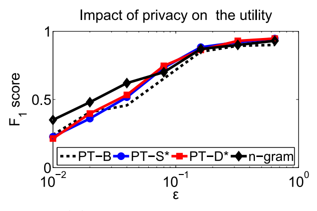


(a) `house_power`

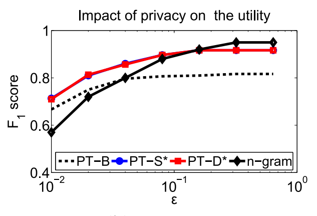


(b) `MSNBC`

Figure 8: Impact of the length of the patterns k=20.



(a) `house_power`



(b) `MSNBC`

Figure 9: Impact of the value of ϵ on the final utility.

result follows the one from the n-gram model. Furthermore, in this dataset we can clearly see the advantages of our techniques compared to the baseline method.

6.2.2 Mining Frequent Prefixes

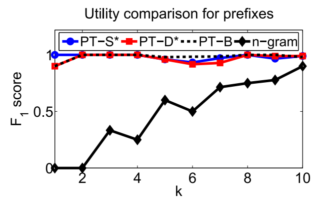
Since by their nature, the frequency of the prefixes are considerably lower than the frequency of substring patterns, it is more reasonable to focus the mining task on short prefixes. In this setting, we examine the performance of the approaches for prefix mining and report the results for both datasets in Figure 10. For both datasets, the n-gram approach provides quite poor utility. The reason is that for this approach the information about the prefix patterns is not preserved. In fact, only substrings are considered in the construction of the n-gram model, and therefore the prefixes cannot be retrieved. On the other hand, in our approaches the use of the first phase allows us to capture these frequent prefix patterns which leads to very accurate results. In a similar way, the baseline PT-B based on the prefix tree structure also achieves good utility in this setting.

7. CONCLUSION

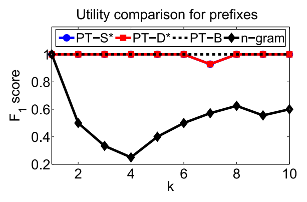
In this paper, we proposed a novel technique for the differentially private mining of frequent sequential patterns. First, we developed two prefix tree mining techniques that make use of statistical information to carefully calibrate the perturbation noise. Second, we introduced a local transformation technique on the original string records which reduces the amount of noise injected by the Laplace Mechanism. We showed how to combine these two key strategies into a novel two-phase mining algorithm. A set of extensive experiments proved that our solutions provide comparable results with the state-of-the-art [9] in mining frequent substrings, while achieving significantly superior results for mining frequent prefixes at the same time. Future research directions include the mining of noisy frequent patterns (e.g. with error and gap), as well as the mining of patterns with concise representation such as closed sequential patterns.

8. ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1117763. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.



(a) `house_power`



(b) `MSNBC`

Figure 10: Comparison for mining prefix patterns, I=[2,3].

9. REFERENCES

[1] O. Abul, F. Bonchi, and M. Nanni. Never walk alone: Uncertainty for anonymity in moving objects databases. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, 2008.

[2] G. Andrienko, N. Andrienko, F. Giannotti, A. Monreale, and D. Pedreschi. Movement data anonymity through generalization. In *Proceedings of the 2nd SIGSPATIAL ACM GIS 2009 International Workshop on Security and Privacy in GIS and LBS*, SPRINGL '09, 2009.

[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, 2007.

[4] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10.

[5] F. Bonchi, L. V. Lakshmanan, and H. W. Wang. Trajectory anonymity in *SIGKDD*, 13(1), Aug. 2011.

[6] L. Bonomi, L. Xiong, R. Chen, and B. C. M. Fung. Frequent grams based embedding for privacy preserving record linkage. In *CIKM*, 2012.

[7] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3), Nov. 2011.

[8] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, 2006.

[9] R. Chen, G. Acs, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, 2012.

[10] R. Chen, B. C. Fung, N. Mohammed, B. C. Desai, and K. Wang. Privacy-preserving trajectory data publishering by local suppression. *Information Sciences*, (0):–, 2011.

[11] R. Chen, B. C. M. Fung, B. C. Desai, and N. M. Sossou. Differentially private transit data publication: A case study on the montreal transportation system. In *Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD).*

[12] C. Dwork. Differential privacy. In *in ICALP*, 2006.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006.*

[14] N. Li, W. Qardaji, D. Su, and J. Cao. Privbasis: frequent itemset mining with differential privacy. *Proc. VLDB Endow.*, 5(11), July 2012.

[15] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3), Jan. 1997.

[16] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, 2009.

[17] M. Rabin. *Fingerprinting by Random Polynomials*. Center for Research in Computing Technology: Center for Research in Computing Technology. 1981.

[18] G. Reinert, S. Schbath, and M. Waterman. Probabilistic and statistical properties of finite words in finite sequences. *Lothaire: Applied Combinatorics on Words*, 2005.

[19] M. Terrovitis and N. Mamoulis. Privacy preservation in the publication of trajectories. In *Proceedings of the The Ninth International Conference on Mobile Data Management*, MDM '08, 2008.

[20] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1), Jan. 1992.

[21] K. Wang, Y. Xu, and J. X. Yu. Scalable sequential pattern mining for biological sequences. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, 2004.

[22] X. Xiao, G. Bender, M. Hay, and J. Gehrke. ireduct: differential privacy with reduced relative errors. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, 2011.

[23] R. Yarovoy, F. Bonchi, L. V. S. Lakshmanan, and W. H. Wang. Anonymizing moving objects: how to hide a mob in a crowd? In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, 2009.

[24] C. Zeng, J. F. Naughton, and J.-Y. Cai. On differentially private frequent itemset mining. *Proc. VLDB Endow.*, 6(1), Nov. 2012.