A Differentially Private Data Structure for Substring and Document Counting

Giulia Bernardini
University of Milan
Milan, Italy
giulia.bernardini@unimi.it

Philip Bille
Technical University of Denmark
Lyngby, Denmark
phbi@dtu.dk

Inge Li Gørtz
Technical University of Denmark
Lyngby, Denmark
inge@dtu.dk

Teresa Anna Steiner
University of Southern Denmark
Odense, Denmark
steiner@imada.sdu.dk

ABSTRACT

For databases consisting of many text documents, one of the most fundamental data analysis tasks is counting (i) how often a pattern appears as a substring in the database (*substring counting*) and (ii) how many documents in the collection contain the pattern as a substring (*document counting*). If such a database contains sensitive data, it is crucial to protect the privacy of individuals in the database. Differential privacy is the gold standard for privacy in data analysis. It gives rigorous privacy guarantees, but comes at the cost of yielding less accurate results.

In this paper, we study the problem of substring and document counting under differential privacy. We give the first differentially private data structures for these problems and provide bounds on their additive error. For ϵ-differential privacy, we show that the error of our data structure is optimal up to a poly-logarithmic factor in the number of documents and length of the longest document. Our data structures immediately lead to improved algorithms for related problems, such as privately mining frequent substrings and q-grams.

1. INTRODUCTION

Differential privacy [14] is the gold standard for privacy in data analysis. It has been extensively studied in theory and employed on a large scale by companies such as Google, Apple, and Uber, as well as public institutions such as the US Census Bureau. The goal of differentially private data analysis is to protect the privacy of any *individual contribution*, while giving information about the *general trends* in the database. This is often achieved as follows: Given a database from which we want to extract some information (specified through a *query*), instead of giving the true output to the query, we add random noise to hide the contribution of any single data point. The random noise is scaled such that for any single data point, any given output should be *roughly equally likely* whether the data point is present in the database or not. Specifically, we say that two databases are *neighboring* if they differ in a single data point. An algorithm is (ϵ, δ)-differentially private if the probabilities of any output differ by at most a factor of e^ϵ and an additive δ between any two neighboring databases. If $\delta = 0$, we call the algorithm ϵ-differentially private. The definition of differential privacy allows us to quantify *how private* the output is. Promising differential privacy naturally comes with a loss of accuracy: a completely random output would be perfectly private, but not very useful. The goal is to achieve a good tradeoff between privacy and accuracy.

Many data analysis applications deal with databases of sequential data, which we refer to as *documents* or *strings*: For example, biological sequence data, web browsing statistics, trajectory data, and text protocols for next-word suggestions. Since these documents contain highly confidential information, an important question is whether such collections can be analyzed while preserving differential privacy. For databases consisting of many documents, we are interested in the *patterns* it contains. In particular: For a database that is a collection of documents $\mathcal{D} = S_1, \ldots, S_n$, and a query pattern P, how many documents in $\mathcal{D}$ contain P? This problem is called Document Count. Another closely related question is: How often does P appear as a substring of a document in $\mathcal{D}$ *in total*? We call this problem Substring Count. The difference between Document Count and Substring Count is that in the latter, if a pattern occurs multiple times in the same document of $\mathcal{D}$, we count all its occurrences, while in the former, any document in $\mathcal{D}$ can contribute at most one to the count of any pattern. In the non-private setting, these problems have been extensively studied from a data-structure perspective, where one preprocesses the collection $\mathcal{D}$ to answer queries for any

This work is supported by the Danish Research Council grant number DFF-8021-002498 and the Villum Fonden grant number VIL51463.

pattern P efficiently.

In this paper, we study the problem of substring and document counting under differential privacy. Specifically, as in the non-private setting, we are interested in designing a data structure for these problems, which can then be queried for an arbitrary number of patterns. For privacy, we want to *keep any document private*, that is, two neighboring databases differ on a single text document. The data structure should output a noisy count for any pattern query, while minimizing the additive error and preserving differential privacy.

Trying to achieve this poses two main challenges. The first challenge is to provide a data structure that can be queried an arbitrary number of times without any privacy loss: Classic differentially private algorithms are designed as one-shot algorithms, that is, we are only allowed to run them once on any given database. Indeed, running several differentially private algorithms on the same database generally results in a higher privacy loss: As an example, consider the case where we want to output how many people in a database satisfy a given property. A differentially private algorithm computes the true output and adds random noise. Now, if we run the same differentially private algorithm on this database many times, then the average of all outputs will converge to the true output plus the mean of the random noise. Thus, at some point, we will be able to give the true answer with high probability, destroying any privacy guarantee. Summarizing, if one were to answer many queries on the same database by just running a new differentially private algorithm each time, then the privacy loss grows with the number of queries.

The second challenge is the high dimensionality of the input data. Recall that our privacy definition requires protecting the influence of any single document. This limits the accuracy we can hope for: As an example, consider the neighboring databases $\mathcal{D}$ and $\mathcal{D}'$, where the length-ℓ string $S_i = aaa\dots a$ of $\mathcal{D}$ is replaced by the length-ℓ string $S_i' = bbb\dots b$ in $\mathcal{D}'$. The substring count of query pattern $P = a$ differs by ℓ between S_i and S_i'. Thus, intuitively, any algorithm computing the count of P which does not significantly distinguish between $\mathcal{D}$ and $\mathcal{D}'$ must have an error of roughly ℓ. Note that this lower bound holds for the problem of computing the count of a single (and very short) pattern! The high dimensionality also introduces another severe challenge: Note that $P = a$ may not even appear in $\mathcal{D}'$ at all - yet it must have a similar probability of producing a positive count as it does in $\mathcal{D}$. This means that assuming our input data consists of strings of a (known) maximum length ℓ, then in order to satisfy ϵ-differential privacy, *any* string of length at most ℓ must have a positive count with some non-zero probability, not only those which actually appear as substrings in $\mathcal{D}$. On the other hand, we would like to efficiently construct a not-too-large data structure, which prohibits explicitly computing a noisy count for all possible patterns.

We tackle the first challenge by giving a *differentially private construction algorithm* for our data structure; since differential privacy is robust to post-processing, this data structure can then be queried ad-libitum without further privacy loss. Thus, preprocessing the database into a data structure has two advantages in this setting: Besides the speedup in queries, which is the usual motivation for data structures, it allows us to avoid the privacy cost of asking many queries. To deal with the high dimensionality of string data, we use insights from string algorithms, along with a filtering of rare patterns at various steps throughout the construction algorithm. We combine this with a novel differentially private algorithm for counting on trees. The result is a differentially private algorithm which produces a trie of $O(n\ell^2)$ nodes, where each node has a noisy count. For ϵ-differential privacy, we show that the maximum error of these noisy counts is $O(\ell)$, up to polylogarithmic factors in the problem parameters, for both Document Count and Substring Count. Strings which are not present in the trie have a count which lies below the error bound, with high probability in the coin-flips of the algorithm. Surprisingly, this almost matches the discussed lower bound for computing the Substring Count of *one* pattern, even though the data structure can be used to compute the count of *all possible* patterns. For (ϵ, δ) differential privacy, we show how to reduce the error for Document Count by a factor $\sqrt{\ell}$.

As an immediate application of our work, we get improved results for problems which have been extensively studied under differential privacy from a practical perspective: namely, the problem of *frequent sequential pattern mining* [7, 29, 11, 27, 24], or the very similar problem of *q-gram extraction* [25, 10], where a q-gram is a pattern of a fixed length q. The goal in both of these problems is to analyze a collection of strings and report patterns that either occur in many strings in the collection (i.e., have a high document count) or appear often as a substring in the collection (i.e., have a high substring count). The private algorithms in these papers have been used for analysing transit data [11] and publishing genome data [24]. However, existing papers do not give any theoretical guarantees on the accuracy of their algorithms in terms of the problem parameters. As an application of our data structures for Document Count or Substring Count, we can solve these problems by reporting all patterns with a noisy count above a given threshold. We show that the additive error achieved by our ϵ-differentially private algorithm is tight for this problem formulation, up to poly-logarithmic factors. Thus, the problem we study is a natural generalization of these problems. In the full version of this paper [5], we argue that our tight bounds yield a polynomial improvement of the error compared to prior approaches.

1.1 Setup and Results

In this work, our database is a collection $\mathcal{D}$ of documents (which we sometimes call strings) of length at most ℓ drawn from an alphabet Σ of size $|\Sigma|$. That is, we have $\mathcal{D} = S_1, \dots, S_n$, where $S_i \in \Sigma^{[1,\ell]}$ for all $i = 1, \dots, n$. Thus, the *data universe* is $X = \Sigma^{[1,\ell]}$ and any database is an element of X^n. We call two databases $\mathcal{D} \in X^n$ and $\mathcal{D}' \in X^n$ *neighboring* if there exists an $S_i' \in \Sigma^{[1,\ell]}$ such that $\mathcal{D}' = S_1, \dots, S_{i-1}, S_i', S_{i+1}, \dots S_n$ for some i (that is, document S_i has been replaced with document S_i'). We also denote the (symmetric) neighboring relation as $\mathcal{D} \sim \mathcal{D}'$.

Definition 1 (Differential Privacy). *An algorithm A with output range* $\mathrm{range}(A)$ *is (ϵ, δ)-differentially private, if for all neighboring $\mathcal{D}$ and $\mathcal{D}'$ and any set $U \subseteq \mathrm{range}(A)$, we have*

$$\Pr[A(\mathcal{D}) \in U] \leq e^{\epsilon} \Pr[A(\mathcal{D}') \in U] + \delta.$$

For $\delta = 0$, the property from Definition 1 is also called ϵ-*differential privacy* or *pure differential privacy*. For $\delta > 0$,

it is also called *approximate differential privacy*.

Let P and S be strings over an alphabet Σ. Given a database $\mathcal{D} = S_1, \ldots, S_n$, we want to build a data structure to efficiently answer the following types of queries while preserving differential privacy:

DOCUMENT COUNT(P)**:** Return the number $\text{count}_1(P, \mathcal{D})$ of documents in $\mathcal{D}$ that contain P at least once.

SUBSTRING COUNT(P)**:** Return the total number $\text{count}(P, \mathcal{D})$ of occurrences of P in the documents in $\mathcal{D}$.

EXAMPLE 1. *Consider the database* $\mathcal{D} = \{$`aaaa, abe, absab, babe, bee, bees`$\}$ *and the string* $P =$ `ab`. *Then* $\text{count}_1(P, \mathcal{D}) = 3$, $\text{count}(P, \mathcal{D}) = 4$.

To ensure privacy, we need to introduce an error. We say that a data structure for DOCUMENT COUNT has an *additive error* α, if for any query pattern $P \in \Sigma^{[1,\ell]}$, it produces a value $\text{count}_1^*(P, \mathcal{D})$ such that $|\text{count}_1(P, \mathcal{D}) - \text{count}_1^*(P, \mathcal{D})| \leq \alpha$. Similarly, a data structure for SUBSTRING COUNT(P) has an *additive error* α, if for any query pattern $P \in \Sigma^{[1,\ell]}$, it produces a value $\text{count}^*(P, \mathcal{D})$ such that $|\text{count}(P, \mathcal{D}) - \text{count}^*(P, \mathcal{D})| \leq \alpha$.

Main results.

In this paper, we give new ϵ-differentially private and (ϵ, δ)-differentially private data structures for DOCUMENT COUNT and SUBSTRING COUNT. We obtain new solutions to frequent substring mining as a corollary. We stress that for all of our data structures, the *construction algorithm* satisfies differential privacy, so the resulting data structures can be queried ad libitum without further privacy loss. We first state our result for ϵ-differential privacy, which gives both a data structure for DOCUMENT COUNT and for SUBSTRING COUNT with an additive error at most $O(\ell \cdot \text{polylog}(n\ell + |\Sigma|))$ with high probability. The data structure output by both Theorems 1 and 2 is a trie with $O(n\ell^2)$ nodes and a noisy count for each node. Patterns which are not present in the trie have a count which lies below the error bound with high probability.

THEOREM 1. *There exists an ϵ-differentially private algorithm, which can process any database $\mathcal{D} = S_1, \ldots, S_n$ of documents in $\Sigma^{[1,\ell]}$ and with high probability outputs a data structure for* SUBSTRING COUNT *(resp.* DOCUMENT COUNT*) with additive error $O\left(\epsilon^{-1} \ell \log \ell (\log^2(n\ell) + \log|\Sigma|)\right)$. The data structure can be constructed in $O(n^2\ell^4)$ time and space, stored in $O(n\ell^2)$ space, and answer queries in $O(|P|)$ time.*

For DOCUMENT COUNT, we can get an (ϵ, δ)-differentially private data structure whose error is better by roughly a factor $\sqrt{\ell}$.

THEOREM 2. *There exists an (ϵ, δ)-differentially private algorithm, which can process any database $\mathcal{D} = S_1, \ldots, S_n$ of documents in $\Sigma^{[1,\ell]}$ and with high probability outputs a data structure for* DOCUMENT COUNT *with additive error $O\left(\epsilon^{-1}\sqrt{\ell \log(1/\delta)} \log \ell \left(\log(n\ell) + \sqrt{\log|\Sigma| \log\log \ell}\right)\right)$. The data structure can be constructed in $O(n^2\ell^4)$ time and space, stored in $O(n\ell^2)$ space, and answer queries in $O(|P|)$ time.*

We can use the data structure to output frequent substrings in a simple way: For a given threshold τ, we traverse the trie and output every string corresponding to a node with a noisy count at least τ. Note that the error of the approximate count propagates to the threshold: Assuming the error of the approximate counts in the trie is at most α, we can say the following: For any string we output, its true count is at least $\tau - \alpha$, and we will find all strings with a count at least $\tau + \alpha$. However, patterns whose count lies within the interval $[\tau - \alpha, \tau + \alpha]$ may be output or not.

In the full version [5], we give a lower bound which states that the bound from Theorem 1 is tight up to $\text{polylog}(n\ell)$ terms. We also give an almost linear time algorithm for computing DOCUMENT COUNT and SUBSTRING COUNT for q-grams with (ϵ, δ)-differential privacy. The error guarantees of that algorithm match the bounds of our main theorems.

In the full version, we also develop a new technique to differentially privately compute count functions on a tree, which may be of independent interest. We motivate this using two examples. First, let T be a tree, where every leaf corresponds to an element in the universe. The count of every leaf is the number of times the element appears in a given database, and the count of a node is the sum of the counts of the leaves below. This model captures any hierarchical composition of data items (i.e., by zip code, area, state), and the problem of computing the node counts has been studied under differential privacy (e.g. [29, 20]). As noted in [20], it can be solved via a reduction to differentially private range counting over the leaf counts. For ϵ-differential privacy, the binary tree mechanism by Dwork et al. [15] gives an error of roughly $O(\log^2 u)$ for this problem, where u is the size of the universe. The range counting problem under (ϵ, δ)-differential privacy has also been extensively studied recently [12, 22, 1, 8, 9, 2]. As a second example, consider again a tree whose leaves are the items of the universe; however, we additionally have a *color* for every item. Now, the goal is to compute for every node the *number of distinct colors* of elements present in the leaves below the node. We call this problem the *colored tree counting problem*. This is similar to the *colored range counting problem*, which is well-studied in the non-private setting [23, 19, 18], and was also recently studied with differential privacy under the name of "counting distinct elements in a time window" [21].

We give an ϵ-differentially private algorithm that can solve the colored tree counting problem with error $O(\log^2 u \log h)$, where h is the height of the tree. In general, our algorithm can construct a differentially private data structure for any counting function that is (i) monotone, in the sense that any node's count cannot be larger than the sum of the counts of its children, and (ii) has bounded L_1-*sensitivity* on the leaves, i.e., the true counts of the leaves do not differ too much on neighboring databases (see Definition 2 for a formal definition of sensitivity). We show an improvement for (ϵ, δ)-differential privacy, if additionally the sensitivity of the count function for every node is small.

1.2 Technical Overview

Simple approach. Before we introduce our algorithm, we present a simple trie-based approach (a similar strategy was adopted in previous work, e.g. [7, 29, 10, 11, 25, 24]). For simplicity, we focus on SUBSTRING COUNT. A private trie for the database is constructed top-down. Starting at the root, we create a new child node for every letter in the alphabet, connected by an edge labeled with the corresponding letter. Then, for each leaf, we compute how many times the corre-

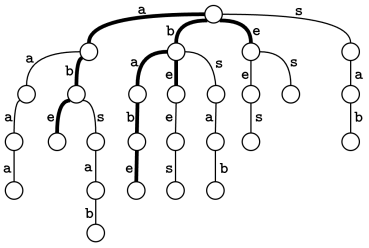

Figure 1: The trie of all suffixes of the strings in database $\mathcal{D}$ from Example 1. The paths corresponding to all suffixes of the string `babe` are in bold.

sponding letter occurs in $\mathcal{D}$ and add noise to this count. If the noisy count exceeds a certain threshold, we expand this node by adding a child for every letter in the alphabet. We continue expanding the trie from the newly added leaves, but in later rounds, when computing counts for a leaf, instead of considering a single letter, we count how many times the string obtained by concatenating the labels of the edges on the root-to-leaf path occurs as a substring in $\mathcal{D}$.

We repeat this process until no noisy count exceeds the threshold. The main advantage of this top-down approach is to avoid considering the entire universe by excluding patterns that are not frequent as soon as a prefix is not frequent. To make the algorithm ϵ-differentially private, the noise we add has to scale approximately with the L_1-sensitivity of the true counts. The L_1-sensitivity is here defined as how much the true counts of the nodes in the trie can differ in total if we replace a document in the database. Assume $\mathcal{D}' = \mathcal{D} \setminus \{S\} \cup \{S'\}$. Then, the count of a node is different on $\mathcal{D}$ and $\mathcal{D}'$ if and only if the string represented by the node is a substring of S or S' (or both). However, S and S' can each have up to $\Omega(\ell^2)$ different substrings. Thus, this approach yields an error of $\Omega(\ell^2)$.

Idea for improvement. An important insight is that while there are $\Omega(\ell^2)$ nodes in the trie that can be influenced by S and S', these nodes can lie on at most 2ℓ different root-to-leaf paths, namely, the paths corresponding to the suffixes of S and S'. See Figure 1. Our idea is to leverage this property. For this, we make use of the *heavy path decomposition* [28] of a tree T, which partitions T into disjoint *heavy paths* and has the property that any root-to-leaf path in T crosses at most a logarithmic number (in the size of T) of such heavy paths. We will use these heavy paths to add noise in a way such that the error depends on the number of heavy paths containing nodes with a count that is different on $\mathcal{D}$ and $\mathcal{D}'$.

However, there are two main challenges. We cannot construct the heavy path decomposition during the top-down construction above, since this requires knowing the size of the subtrees before they have been explored, but at the same time, we need the noisy counts to decide which nodes to expand further. Alternatively, we could first construct the full trie of the universe (all possible substrings of length ℓ) and then compute the heavy path decomposition. But in this case, the size of the trie would be $|\Sigma|^\ell$, so the number of heavy paths on a root-to-leaf path would only be bounded by $\ell \log \Sigma$, and therefore, the ℓ root-to-leaf paths which contain nodes of different counts in $\mathcal{D}$ and $\mathcal{D}'$ can cross up to $\Omega(\ell^2 \log \Sigma)$ heavy paths.

Our algorithm. We now give an overview of our full algorithm. As a first step, we reduce the universe size from $|\Sigma|^\ell$ to $n^2\ell^3$, by computing a set of *candidate frequent strings* in a differentially private way. Recall that we need substrings not occurring in our database $\mathcal{D}$ to appear in our candidate set with non-zero probability. We show how to efficiently obtain this without exhaustively considering all such substrings.

We iteratively construct a collection $\mathcal{P} = \mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^{\lfloor \log \ell \rfloor}}$ of sets of strings, such that the strings in $\mathcal{P}_{2^k}$ have length 2^k, as follows. We first construct $\mathcal{P}_{2^0}$ by computing noisy counts for all letters in the alphabet, and keep only those with a noisy count above a certain threshold. To construct $\mathcal{P}_{2^k}$, we compute noisy counts of all strings obtained by concatenating two strings from $\mathcal{P}_{2^{k-1}}$, again keeping only those with a noisy count above the threshold. We show that the collection only contains substrings from $\mathcal{D}$ with high probability. Note that a document S can have at most ℓ substrings of any fixed length, and we consider a logarithmic number of lengths. Therefore, our noise needs to scale with $\ell \log \ell$. We then use the following observation: any string of a given length m which is not a power of two has an overlapping prefix and suffix whose length is a power of two. Our final candidate set consists of the collection $\mathcal{P}$ together with all strings covered by a prefix and suffix from $\mathcal{P}$.

Next, we build a trie from our candidate set and decompose it into heavy paths. For every heavy path, we compute (i) a differentially private noisy count of its root, and (ii) a differentially private estimate of the *prefix sums of the difference sequence of the counts* on the path going down. Specifically, for every node on the heavy path, we get an approximation of the *difference* between its count and the count of the root. We can compute (ii) with low noise using a generalized variant of the binary tree mechanism [15]. We show that since any root-to-leaf path can cross at most $\log(n^2\ell^3) = O(\log(n\ell))$ heavy paths, and a document S can only influence counts on the at most ℓ paths corresponding to its suffixes, this can be done with an error which is $O(\ell \cdot \text{polylog}(n\ell))$. This strategy gives our main result of Theorem 1.

The improvement for q-grams stems from two observations: 1) since we only want to compute the counts for patterns of a fixed length, we do not need to compute the full set of candidate strings and can significantly simplify the algorithm, and 2) for (ϵ, δ)-differential privacy, we show that we can avoid computing noisy counts for patterns whose true count is 0, without violating privacy.

1.3 Roadmap

In Section 2, we collect some background on string data structures and differential privacy. In Section 3, we present the construction algorithm for our ϵ-differentially private data structure for Substring Count. In Section 4, we sketch a faster construction algorithm for Substring Count restricted to q-grams with (ϵ, δ) differential privacy.

2. PRELIMINARIES

In this work, we use log to denote the binary logarithm and ln to denote the natural logarithm. We use $[a, b]$ to denote the interval of integers $\{a, a+1, a+2, \ldots, b-1, b\}$.

A string S of length $|S| = \ell$ is a sequence $S[0] \cdots S[\ell-1]$ of ℓ characters drawn from an alphabet Σ of size $|\Sigma|$. The string $S[i] \cdots S[j]$, denoted $S[i,\ j]$, is called a *substring* of

S; $S[0,\ j-1]$ and $S[i,\ \ell-1]$ are a *prefix* and *suffix* of S, respectively. We say that a string P *occurs* in a string S iff there exists an i such that $S[i,\ i+|P|-1] = P$. We use $\Sigma^{[a,b]}$ to denote all strings S over Σ which satisfy $a \leq |S| \leq b$.

A *trie* for a collection of strings $\mathcal{C} = S_1, \ldots, S_n$, denoted $T_{\mathcal{C}}$, is a rooted labeled tree, such that: (1) The label on each edge is a character of one or more S_i. (2) Each string in $\mathcal{C}$ is represented by a path in $T_{\mathcal{C}}$ going from the root down to some node (obtained by concatenating the labels on the edges of the path). (3) Each root-to-leaf path represents a string from $\mathcal{C}$. (4) Common prefixes of two strings share the same path maximally. For a node $v \in T_{\mathcal{C}}$, we let $\mathrm{str}(v)$ denote the string obtained from concatenating the labels on the path's edges from the root to v.

A *compacted trie* is obtained from $T_{\mathcal{C}}$ by dissolving all nodes except the root, the branching nodes, and the leaves, and concatenating the labels on the edges incident to dissolved nodes to obtain *string* labels for the remaining edges. The compacted trie can be represented in $O(|\mathcal{C}|)$ space and computed in $O(|\mathcal{C}|)$ time [26].

Definition 2 (L_p-sensitivity). *Let f be a function $f : \chi^n \to \mathbb{R}^k$ for some universe χ. The L_p-*sensitivity of f *is defined as $\max_{\mathcal{D}\sim\mathcal{D}'} ||f(\mathcal{D}) - f(\mathcal{D}')||_p$.*

The sensitivity of a function is a measure of how much its output can differ between neighboring databases. The *Laplace Mechanism* [14] computes the output of a function plus Laplace noise with a scale depending on ϵ and the L_1-sensitivity of the function. Its properties are summarized in the following corollary.

Corollary 1. *Let f be a function $f : \chi^n \to \mathbb{R}^k$ for some universe χ with L_1-sensitivity at most Δ_1. Then there exists an ϵ-differentially private algorithm A which for any $\mathcal{D} \in \chi^n$ outputs $A(\mathcal{D})$ satisfying $||A(\mathcal{D}) - f(\mathcal{D})||_\infty \leq \epsilon^{-1}\Delta_1 \ln(k/\beta)$ with probability at least $1-\beta$.*

Lemma 1 is due to Dwork and Lei [13] and shows how privacy degrades when combining multiple differentially private algorithms. For ϵ-differential privacy, this composition lemma is tight in the worst case (for (ϵ, δ)-differential privacy, tighter results are known, but would only save logarithmic factors in this work).

Lemma 1. *Let A_1 be an (ϵ_1, δ_1)-differentially private algorithm $\chi^n \to \mathrm{range}(A_1)$ and A_2 an (ϵ_2, δ_2)-differentially private algorithm $\chi^n \times \mathrm{range}(A_1) \to \mathrm{range}(A_2)$. Then $A_1 \circ A_2$ is $(\epsilon_1+\epsilon_2, \delta_1+\delta_2)$-differentially private.*

3. CONSTRUCTION ALGORITHM

In this section, we show how to construct our data structure. For simplicity, we focus on Substring Count. The solution for Document Count is similar. We make the following key observation.

Observation 1. *For any document $S \in \mathcal{D}$ and any length $m \leq \ell$, the cumulative count $\sum_{P\in\Sigma^m} \mathrm{count}(P, S)$ of all length-m substrings of S is at most ℓ.*

This observation holds because, since the length of S is bounded by ℓ, S has at most $\ell - m + 1 \leq \ell$ fragments of length m (one for each possible starting position). Thus the total number of occurrences of its substrings of length m is bounded by the same quantity. Observation 1 implies the following.

Corollary 2. *The L_1-sensitivity of $(\mathrm{count}(P, \mathcal{D}))_{P\in\Sigma^m}$ is bounded by 2ℓ, for any $m \leq \ell$.*

Our algorithm has 6 main steps.

Step 1. We run an ϵ-differentially private algorithm to compute a set $\mathcal{C}$ of *candidate frequent* strings such that the true count in $\mathcal{D}$ of any string not included in $\mathcal{C}$ is small, and the set $\mathcal{C}$ is not too large.

Step 2. We build the trie $T_{\mathcal{C}}$ of the candidate set $\mathcal{C}$, and construct a *heavy path decomposition* of $T_{\mathcal{C}}$.

Step 3. For every root r of a heavy path, we compute a differentially private estimate of $\mathrm{count}(\mathrm{str}(r), \mathcal{D})$.

Step 4. For every heavy path $p = v_0, v_1, \ldots, v_{|p|-1}$ with root $r = v_0$, we consider the *difference sequence* of counts along the path: i.e., for a node v_i on the path and its parent v_{i-1}, the ith element in the difference sequence is given by $\mathrm{count}(\mathrm{str}(v_i), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v_{i-1}), \mathcal{D})$, for $i = 1, \ldots, |p| - 1$. We use the binary tree mechanism [15] to compute a differentially private estimate of all *prefix sums*, sums^*_p, of each of the difference sequences.

Step 5. We compute the noisy count of every node v in $T_{\mathcal{C}}$, thus of every substring in $\mathcal{C}$: For every node v_i on a heavy path $v_0, v_1, \ldots, v_{|p|-1}$, we set $\mathrm{count}^*(\mathrm{str}(v_i), \mathcal{D}) = \mathrm{count}^*(\mathrm{str}(v_0), \mathcal{D}) + \mathrm{sums}^*_p[i]$.

Step 6. We finally prune $T_{\mathcal{C}}$ removing the subtrees with sufficiently small noisy counts, and return the pruned trie as our ϵ-differentially private data structure.

In the rest of the section, we detail this approach.

3.1 Step 1: Computing a Candidate Set

As a first step, we show how to compute a candidate set $\mathcal{C}$ satisfying the following lemma.

Lemma 2. *Let $\mathcal{D} = S_1, \ldots, S_n$ be a database of documents. For any $\epsilon > 0$ and $0 < \beta < 1$ there exists an ϵ-differentially private algorithm, which computes a candidate set $\mathcal{C} \subseteq \Sigma^{[1,\ell]}$ enjoying the following two properties with probability at least $1-\beta$:*

- $\mathrm{count}(P, \mathcal{D}) = O(\epsilon^{-1}\ell \log \ell \log(\max\{\ell^2 n^2, |\Sigma|\}/\beta))$ *for any $P \in \Sigma^{[1,\ell]}$ not in $\mathcal{C}$*
- $|\mathcal{C}| \leq n^2\ell^3$

We compute the candidate set as follows.

1.1 Inductively construct sets $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$, with $j = \lfloor \log \ell \rfloor$, where $\mathcal{P}_{2^k}$ contains only strings of length 2^k which appear sufficiently often as substrings in $\mathcal{D}$ according to noisy counts;

1.2 Compute, for each $m \leq \ell$ which is not a power of 2, the set $\mathcal{C}_m$ of all strings P of length m whose length-2^k prefix and suffix are in $\mathcal{P}_{2^k}$, where $k = \lfloor \log m \rfloor$;

1.3 The set $\mathcal{C}$ is finally obtained as the union of all the sets computed in Steps 1.1 and 1.2.

More precisely, in Step 1.1 the algorithm first computes a noisy count of all letters via the Laplace mechanism (Corollary 1). From those, it computes the set $\mathcal{P}_{2^0}$ of letters whose noisy counts are above a threshold τ which is strictly larger

than the error bound of the Laplace mechanism.Thus, with high probability, we only keep letters appearing in $\mathcal{D}$, i.e., at most $n\ell$. It then constructs each of the sets $\mathcal{P}_{2^k}$ from $\mathcal{P}_{2^{k-1}}$ by computing noisy counts for all strings obtained by concatenating two strings from $\mathcal{P}_{2^{k-1}}$ and pruning those with a noisy count lower than τ. We maintain $|\mathcal{P}_{2^k}| \leq n\ell$ for all k.

EXAMPLE 2. *Consider the database* $\mathcal{D} = \{$`aaaa, abe, absab, babe, bee, bees`$\}$ *from Example 1. The exact versions,* $\mathcal{P}^{\times}_{2^0}$, $\mathcal{P}^{\times}_{2^1}$, $\mathcal{P}^{\times}_{2^2}$, *of the sets* $\mathcal{P}_{2^0}$, $\mathcal{P}_{2^1}$, $\mathcal{P}_{2^2}$ *(i.e. the sets computed according to exact rather than noisy counts) with a frequency threshold* $\tau = 1$ *are*

$$\mathcal{P}^{\times}_{2^0} = \{\texttt{a}, \texttt{b}, \texttt{e}, \texttt{s}\}$$
$$\mathcal{P}^{\times}_{2^1} = \{\texttt{aa}, \texttt{ab}, \texttt{ba}, \texttt{be}, \texttt{bs}, \texttt{ee}, \texttt{es}, \texttt{sa}\}$$
$$\mathcal{P}^{\times}_{2^2} = \{\texttt{aaaa}, \texttt{absa}, \texttt{babe}, \texttt{bees}, \texttt{bsab}\}$$

The noisy, ϵ*-differentially private sets* $\mathcal{P}_{2^0}$, $\mathcal{P}_{2^1}$, $\mathcal{P}_{2^2}$ *that are actually computed by the construction algorithm may contain (with sufficiently low probability) substrings that do not occur at all in* $\mathcal{D}$*; or conversely, some substrings occurring in* $\mathcal{D}$ *may be missing from the sets. A possible outcome of a noisy computation of these sets for* $\mathcal{D}$ *is*

$$\mathcal{P}_{2^0} = \{\texttt{a}, \texttt{b}, \texttt{e}, \texttt{s}\}$$
$$\mathcal{P}_{2^1} = \{\texttt{aa}, \texttt{ab}, \texttt{ba}, \texttt{be}, \texttt{bs}, \texttt{ee}, \texttt{sa}\}$$
$$\mathcal{P}_{2^2} = \{\texttt{aaaa}, \texttt{absa}, \texttt{babe}, \texttt{bsab}, \texttt{aaab}\}$$

Note that the string `es` *is missing from* $\mathcal{P}_{2^1}$ *although it occurs once in* $\mathcal{D}$*, and consequently,* `bees` *is missing from* $\mathcal{P}_{2^2}$*; conversely,* `aaab` *is in* $\mathcal{P}_{2^2}$ *although it does not occur in* $\mathcal{D}$*.*

The noisy counts for the strings in $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$ can be obtained via the Laplace mechanism from their exact counts, which in turn, can be computed efficiently by building a concatenation data structure [6] over the suffix tree of the strings in $\mathcal{D}$ [17]. Details are in [5].

Step 1.2, that is, computing candidate sets for strings whose lengths are not powers of 2, can be done as a post-processing step using sets $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$, thus it does not entail any further privacy loss. For any $2^k < m < 2^{k+1}$, the set $\mathcal{C}_m$ contains all strings of length m having their length-2^k prefix and suffix in $\mathcal{P}_{2^k}$, regardless of whether they occur in $\mathcal{D}$. Note that $|\mathcal{C}_m| \leq n^2\ell^2$ for each m.

EXAMPLE 3. *Consider the same database* $\mathcal{D}$ *and sets* $\mathcal{P}_{2^0}$, $\mathcal{P}_{2^1}$, $\mathcal{P}_{2^2}$ *as in Example 2. The set* $\mathcal{C}_3$ *is*

$$\mathcal{C}_3 = \{\texttt{aaa}, \texttt{aab}, \texttt{aba}, \texttt{abe}, \texttt{abs}, \texttt{baa}, \texttt{bab}, \texttt{bee}, \texttt{bsa}, \texttt{eee}, \texttt{saa}, \texttt{sab}\}$$

$\mathcal{C}_3$ *contains all possible strings of length* 3 *whose length-*2 *prefix and suffix are in* $\mathcal{P}_{2^1}$*; these strings can be obtained by taking all ordered pairs of strings* $Q_1, Q_2 \in \mathcal{P}_{2^1}$ *such that the length-*1 *suffix of* s_1 *is equal to the length-*1 *prefix of* s_2*: in this case, we say that* Q_1 *and* Q_2 *have a suffix/prefix overlap of length* 1*. Note that many strings of* $\mathcal{C}_3$ *(e.g.* `aba`*) do not occur at all in* $\mathcal{D}$ *although, in this example, all strings of* $\mathcal{P}_{2^1}$ *do. This is a normal behavior of our construction algorithm, and it is crucial to guarantee* ϵ*-differential privacy.*

The set $\mathcal{C}_5$ *is obtained from* $\mathcal{P}_{2^2}$ *by taking all ordered pairs of strings with a suffix/prefix overlap of length* 3*: we have*

$$\mathcal{C}_5 = \{\texttt{aaaaa}, \texttt{aaaab}, \texttt{absab}\}$$

In general, the strings of a set $\mathcal{C}_m$ with $2^k < m < 2^{k+1}$ can be obtained by computing all pairs $Q_1, Q_2 \in \mathcal{P}_{2^k}$ such that the length-$(2^{k+1} - m)$ suffix of Q_1 is equal to the length-$(2^{k+1} - m)$ prefix of Q_2. These pairs can be computed efficiently by constructing an LCE data structure [4]: details are in [5].

The complete candidate set $\mathcal{C}$ is finally obtained as the union of the sets $\mathcal{P}_{2^j}$ and $\mathcal{C}_m$ constructed in Steps 1.1 and 1.2. The above algorithm implies Lemma 3.

LEMMA 3. *Given* $\mathcal{D} = S_1, \ldots, S_n$ *a database over* $\Sigma^{[1,\ell]}$*, a set* $\mathcal{C}$ *satisfying the properties of Lemma 2 can be computed in time* $O\left(n^2\ell^3 \log\log(n\ell) + n^2\ell^4\right)$ *and space* $O(n^2\ell^4)$*.*

3.2 Steps 2 to 4: Heavy Paths and Difference Sequences

The next steps are to arrange $\mathcal{C}$ in a trie $T_{\mathcal{C}}$ and compute its heavy path decomposition. Note that the number of nodes $|T_{\mathcal{C}}|$ is bounded by the total length of strings in $\mathcal{C}$, which is in turn bounded by $n^2\ell^4$. This step of the algorithm exploits the way the counts associated with nodes of $T_{\mathcal{C}}$ can vary in neighboring databases. A first key observation is as follows: Consider the counts of the string $\mathrm{str}(v)$ associated with any node v of $T_{\mathcal{C}}$ in two neighboring databases $\mathcal{D}$ and $\mathcal{D}' = \mathcal{D} \setminus \{S\} \cup \{S'\}$. The difference of the counts of $\mathrm{str}(v)$ in $\mathcal{D}$ and $\mathcal{D}'$ depends only on how many times $\mathrm{str}(v)$ occurs in the strings S and S' by which $\mathcal{D}$ and $\mathcal{D}'$ differ:

OBSERVATION 2. *For any node* v *in* $T_{\mathcal{C}}$ *and any two neighboring databases* $\mathcal{D}$ *and* $\mathcal{D}' = \mathcal{D} \setminus \{S\} \cup \{S'\}$, $|\mathrm{count}(\mathrm{str}(v), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v), \mathcal{D}')| = |\mathrm{count}(\mathrm{str}(v), S) - \mathrm{count}(\mathrm{str}(v), S')|$.

Let us now focus on how the counts computed in neighboring databases can vary along paths of $T_{\mathcal{C}}$. For any path $p = v_0, v_1, \ldots, v_{|p|-1}$ in the trie $T_{\mathcal{C}}$, we define the *difference sequence of* count *on* p as the $(|p| - 1)$-dimensional vector $\mathrm{diff}_p(\mathcal{D})[i] = \mathrm{count}(\mathrm{str}(v_i), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v_{i-1}), \mathcal{D})$ for $i = 1 \ldots |p| - 1$: see Figure 3(a) for an example. Note that the counts associated to the nodes of a path are non-increasing when descending the path: this is because, if v_j is a descendant of v_i, then $\mathrm{str}(v_i)$ is a prefix of $\mathrm{str}(v_j)$, thus $\mathrm{count}(\mathrm{str}(v_i), S) \geq \mathrm{count}(\mathrm{str}(v_j), S)$. This allows us to prove Lemma 4 (the proof is in [5]).

LEMMA 4. *Let* $\mathcal{D}$ *and* $\mathcal{D}' = \mathcal{D} \setminus \{S\} \cup \{S'\}$ *be two neighboring databases. Then, for any path* p *with root* r *in* $T_{\mathcal{C}}$, $||\mathrm{diff}_p(\mathcal{D}) - \mathrm{diff}_p(\mathcal{D}')||_1 \leq \mathrm{count}(\mathrm{str}(r), S) + \mathrm{count}(\mathrm{str}(r), S')$.

We now pair these properties with the well-known *heavy path decomposition* of a tree T, defined as follows. Every edge is either *light* or *heavy*. There is exactly one heavy edge outgoing from every node except the leaves, defined as the edge to the child whose subtree contains the most nodes; ties are broken arbitrarily. The longest *heavy path* is obtained by following the heavy edges from the root of T to a leaf: note that all edges hanging off this path are light. The other heavy paths are obtained by recursively decomposing all subtrees hanging off a heavy path. We call the topmost node of a heavy path (i.e., the only node of the path which is not reached by a heavy edge) its *root*. The heavy path decomposition of a tree with N nodes can be constructed in $O(N)$ time and has the following important property [28]:

LEMMA 5. *Given a tree* T *with* N *nodes and a heavy path decomposition of* T*, any root-to-leaf path in* T *contains at most* $\lceil \log N \rceil$ *light edges.*

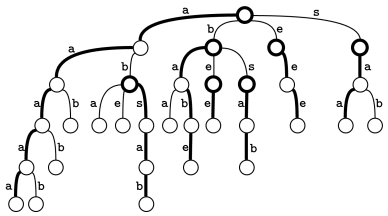

Figure 2: The trie of the candidate set from Example 3. Bold edges are heavy; bold nodes are the roots of the heavy paths.

EXAMPLE 4. *Consider the candidate set* $\mathcal{C} = \mathcal{P}_{2^0} \cup \mathcal{P}_{2^1} \cup \mathcal{P}_{2^2} \cup \mathcal{C}_3 \cup \mathcal{C}_5$ *computed in Examples 2 and 3. The trie* $T_{\mathcal{C}}$ *of* $\mathcal{C}$ *with its heavy path decomposition is depicted in Figure 2.*

We now exploit Lemma 5, together with the fact that any string S of length at most ℓ (not necessarily included in the database from which $T_{\mathcal{C}}$ has been constructed) can only influence counts over the at most ℓ paths of $T_{\mathcal{C}}$ corresponding to its suffixes, to arrive at Lemma 6.

LEMMA 6. *Let* $r_0, r_1, \ldots, r_k$ *be the roots of the heavy paths of* $T_{\mathcal{C}}$*, where* r_0 *is the root of* $T_{\mathcal{C}}$*. Then for any string* S *of length at most* ℓ*, we have* $\sum_{i=0}^{k} \mathrm{count}(\mathrm{str}(r_i), S) \leq \ell(\lceil \log |T_{\mathcal{C}}| \rceil + 1) = O(\ell \log(n\ell))$.

Observation 2 and Lemma 6 give that the L_1-sensitivity of $(\mathrm{count}(\mathrm{str}(r_i), \mathcal{D}))_{i=0}^{k}$ is bounded by $\ell(\lceil \log |T_{\mathcal{C}}| \rceil + 1) = O(\ell \log(n\ell))$. Additionally note that, since $T_{\mathcal{C}}$ has at most $n^2\ell^3$ leaves, there are at most $n^2\ell^3$ heavy paths. Together with Corollary 1, this gives an ϵ-differentially private algorithm to compute noisy counts for the heavy paths roots:

COROLLARY 3. *Let* $r_0, \ldots, r_k$ *be the roots of the heavy paths of* $T_{\mathcal{C}}$*. For any* $\epsilon > 0$ *and* $0 < \beta < 1$*, there exists an* ϵ*-differentially private algorithm to estimate* $\mathrm{count}(\mathrm{str}(r_i), \mathcal{D})$ *up to an additive error at most* $O(\epsilon^{-1}\ell \log(n\ell) \ln(k/\beta)) = O(\epsilon^{-1}\ell \log^2(n\ell/\beta))$ *with probability at least* $1 - \beta$*.*

Additionally, Lemmas 4 and 6 give that the *sum of* L_1-sensitivities of the difference sequences *over all heavy paths* is bounded by $2\ell(\lceil \log |T_{\mathcal{C}}| \rceil + 1)$. As the next step, we show how to privately compute all *prefix sums* of the difference sequences of all heavy paths with an error $O(\ell \lceil \log |T_{\mathcal{C}}| \rceil)$ up to poly-logarithmic terms. This is achieved using a variant of the binary tree mechanism [15] on the difference sequence of each heavy path. The binary tree mechanism works as follows: To compute a differentially private estimate of all prefix sums of a finite sequence, we first divide the sum into partial sums of power-of-two many elements, and add noise to each partial sum. That is, to compute all prefix sums of a sequence $a_1, \ldots, a_Q$, we first compute a noisy value of each a_i, then of $a_1 + a_2$, $a_3 + a_4$, ..., then of $a_1 + a_2 + a_3 + a_4$, $a_5 + a_6 + a_7 + a_8$, and so on. The main idea is that any value a_i can influence at most $\log Q$ of these partial sums, and we can compute an estimate of any prefix sum by summing at most $\log Q$ of the noisy partial sums. We describe this approach in detail in [5, Lemma 3.9].

A schema of the application of Corollary 4 to one of the heavy paths of the trie of Figure 2 is in Figure 3.

COROLLARY 4. *Let* $p_0, \ldots, p_k$ *be all heavy paths of* $T_{\mathcal{C}}$*. For any* $\epsilon > 0$ *and* $0 < \beta < 1$*, there exists an* ϵ*-differentially private algorithm which estimates* $\sum_{j=1}^{i} \mathrm{diff}_{p_m}(\mathcal{D})[j]$ *for all* $i = 1, \ldots, |p_m|$ *and all* $m = 0, \ldots, k$ *up to an additive error at most*

$$O\left(\epsilon^{-1}\ell \log(n\ell) \log \ell \log(\ell k/\beta)\right) = O\left(\epsilon^{-1}\ell \log \ell \log^2(n\ell/\beta)\right)$$

with probability at least $1 - \beta$*.*

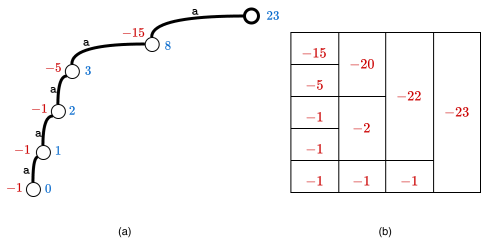

Figure 3: (a) The heavy path rooted at the root of the trie of Figure 2; the real counts are in blue, the difference sequence is in red. The root represents the empty string, assumed to occur at every position of the database $\mathcal{D}$ from Example 3. (b) The partial sums of the difference sequence (without noise). The algorithm computes a noisy version of the values in the table shown.

Combining the construction algorithm of Section 3.1 with the results of this section, we obtain the following.

LEMMA 7. *Given the representation of* $\mathcal{C}$ *output by the algorithm of Section 3.1 for a database* $\mathcal{D}$ *of* n *documents from* $\Sigma^{[1,\ell]}$*, a trie* $T_{\mathcal{C}}$ *and its heavy path decomposition, noisy counts of the heavy path roots and noisy prefix sums on the heavy paths can be computed in* $O(n^2\ell^4)$ *time and space.*

3.3 Steps 5 and 6: Putting Everything Together

Finally, we compute the noisy counts for each node and prune the trie as follows. For a heavy path $p = v_0, v_1, \ldots, v_{|p|-1}$ with root $r = v_0$, let $\mathrm{count}^*(\mathrm{str}(r), \mathcal{D})$ be the noisy count computed for the root, and let $\mathrm{sums}_p^*[i]$ be the noisy estimate of $\sum_{j=1}^{i} \mathrm{diff}_p(\mathcal{D})[j]$. Then for any node v_i on the path with $i > 0$, we compute $\mathrm{count}^*(\mathrm{str}(v_i), \mathcal{D}) = \mathrm{count}^*(\mathrm{str}(r), \mathcal{D}) + \mathrm{sums}_p^*[i]$.

Parameters. To obtain an ϵ-differential private algorithm, we set $\epsilon' = \epsilon/3$ and failure probability $\beta' = \beta/3$ in Steps 1, 3, and 4 of the algorithm.

Let α be the sum of the error bounds for Corollaries 3 and 4 which each hold with probability $1 - \beta'$. We have $\alpha = O(\epsilon^{-1}\ell \log \ell \log^2(n\ell/\beta))$. As a final step, we prune $T_{\mathcal{C}}$ by traversing the trie from the root, and for any node v with $\mathrm{count}^*(\mathrm{str}(v_i), \mathcal{D}) < 2\alpha$, we delete v and its subtree. We return the resulting pruned trie T^* together with noisy counts $\mathrm{count}^*(\mathrm{str}(v), \mathcal{D})$ for every node $v \in T^*$.

By the composition theorem (Lemma 1), the whole algorithm is ϵ-differentially private. By a union bound, all its error guarantees hold together with probability $1 - 3\beta' = 1 - \beta$. Thus, we get with probability $1 - \beta$ a trie T^*, with the following properties.

- For each node $v \in T^*$, by Corollaries 3 and 4,

$$|\mathrm{count}^*(\mathrm{str}(v), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v), \mathcal{D})| \leq \alpha = O\left(\epsilon^{-1}\ell \log \ell \log^2(n\ell/\beta)\right)$$

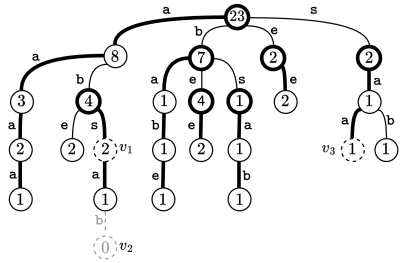

Figure 4: A possible output of the construction algorithm for the database of Examples 2 and 3, obtained by pruning the trie in Figure 2 according to noisy counts. Each node contains its noisy count. Dashed nodes contain erroneous counts: the exact count of v_1 is 1; v_2 has been pruned because of the noise, but its exact count is 1; the exact count of v_3 is 0, but it has not been pruned due to the added noise.

- Every string $P \in \Sigma^{[1,\ell]}$ which is not present T^* was either: (i) not present in $\mathcal{C}$, in which case $\text{count}(P,\mathcal{D}) = O(\epsilon^{-1}\ell \log \ell(\log(n\ell/\beta) + \log|\Sigma|))$ by Lemma 2; or (ii) deleted in the pruning process of $T_{\mathcal{C}}$, in which case $\text{count}(P,\mathcal{D}) < 3\alpha = O(\epsilon^{-1}\ell \log \ell \log^2(n\ell/\beta))$.
- Any string $P \in \sum^{[1,\ell]}$ in T^* has a count at least $2\alpha - \alpha > 1$. Therefore, T^* has at most $O(n\ell^2)$ nodes.

To query the resulting data structure for a pattern P, we match P in the trie, and if there is a node v with $\text{str}(v) = P$, we return the approximate count $\text{count}^*(\text{str}(v),\mathcal{D})$ saved at v. If P is not present, we return 0. This requires $O(|P|)$ time. In summary, we have obtained Theorem 1.

4. EFFICIENTLY COUNTING Q-GRAMS

In this section, we give an intuition of why an (ϵ,δ)-differentially private data structure that answers q-gram counting queries in optimal time matching the error bounds of Theorem 2 (for Document Count) and those of Theorem 1 (for Substring Count) can be constructed more efficiently than the general ϵ-differentially private data structure of Section 3. Specifically, we show how such a data structure can be computed in $O(n\ell(\log q + \log|\Sigma|))$ time and $O(n\ell)$ space:

Similar to the algorithm of Section 3, the construction algorithm restricted to q-grams begins by computing sets $\mathcal{P}_{2^0}, \mathcal{P}_{2^1}, \ldots, \mathcal{P}_{2^j}$, with $j = \lfloor \log q \rfloor$, containing substrings of geometrically increasing length with sufficiently high noisy counts in $\mathcal{D}$. However, for the less stringent model of approximate differential privacy, we crucially observe that we can avoid considering substrings that do not appear in $\mathcal{D}$.

As a consequence, sets $\mathcal{P}_{2^0}, \mathcal{P}_{2^1}, \ldots, \mathcal{P}_{2^j}$ can be computed by filtering the substrings occurring in $\mathcal{D}$ according to noisy counts, rather than generating a quadratic number of candidate strings at every step. This removes the computational bottleneck of the algorithm of Section 3.

The algorithm proceeds inductively starting from $\mathcal{P}_{2^0}$: It computes noisy counts for all letters of Σ occurring in $\mathcal{D}$ at least once, and filters out those whose noisy count is below an appropriately chosen threshold τ. For all subsequent sets $\mathcal{P}_{2^k}$, it computes noisy counts for all length-2^k strings that occur in $\mathcal{D}$ and whose length-2^{k-1} prefix and suffix are in $\mathcal{P}_{2^{k-1}}$. Finally, it computes the set $\mathcal{P}_q$ of strings occurring in $\mathcal{D}$ whose length-2^j prefix and suffix are in $\mathcal{P}_{2^j}$ and whose noisy counts exceed τ, and returns a compacted trie of $\mathcal{P}_q$ with a noisy counter for each node.

To perform these steps efficiently, we rely on the suffix tree of $\mathcal{D}$ [16] preprocessed to answer weighted ancestor queries in $O(1)$ time [3], and we represent sets $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$ implicitly in the suffix tree. Details are in [5].

5. CONCLUSION AND DISCUSSION

In this paper, we present the first differentially private data structures for two fundamental pattern matching problems, Document Count and Substring Count. We show upper bounds on the error for these problems and complement them with lower bounds, showing that the accuracy of our data structures is tight up to polylog$(n\ell)$ terms. We extend this to get improved differentially private algorithms for frequent pattern mining and frequent q-gram mining. Finally, we present new data structures for counting on trees.

To achieve our results, we have developed several new techniques and combinations thereof for private data structures on strings and trees, including a new way to compute counting functions on tree structures using tree heavy path decomposition combined with the binary tree mechanism. Since such counting problems are basic primitives in computer science, our techniques are likely applicable in a wide range of areas. For example, possible applications could include string indexing for complex patterns (e.g., multiple patterns, regular expressions, and patterns with wildcards) and geometric data structure problems such as higher-dimensional range searching.

A key feature of our result is that we construct a differentially private data structure that we can ask many queries on without incurring further loss of privacy. While some differentially private data structures are known, we believe there is significant unexplored potential in designing differentially private solutions for classic data structure problems.

Our error bounds for pattern matching in the ϵ-differential privacy setting depend linearly on the string lengths and logarithmically on the number of strings. Thus, we expect our approach to perform well on databases containing many short or semilong strings, e.g., next-word suggestions or transit data. On the other hand, in scenarios with very long strings, e.g., genomic sequences, the error may be too large for practical applications. However, by our lower bound, this is unavoidable for ϵ-differential privacy if we want to protect entire strings. It is an interesting open question whether our error bounds for approximate differential privacy can be improved or shown to be tight.

For large databases, our construction time and space for general counting on strings may be infeasible. For q-grams, however, the construction time is almost linear. It is a relevant open problem whether the construction time and space can be improved in the general case.

In our presentation, we have focused on simplicity, and we have not attempted to optimize constants in our approach. However, when implementing our algorithms, one can likely obtain better error guarantees, e.g., by allocating the privacy budget differently across the algorithm's components.

6. REFERENCES

[1] N. Alon, R. Livni, M. Malliaris, and S. Moran. Private PAC learning implies finite littlestone dimension. In *Proc. 51st STOC*, pages 852–860, 2019.

[2] A. Beimel, K. Nissim, and U. Stemmer. Private learning and sanitization: Pure vs. approximate differential privacy. *Theory Comput.*, 12(1):1–61, 2016.

[3] D. Belazzougui, D. Kosolobov, S. J. Puglisi, and R. Raman. Weighted ancestors in suffix trees revisited. In *Proc. 32nd CPM*, volume 191 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[4] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proc. 4th LATIN*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000.

[5] G. Bernardini, P. Bille, I. L. Gørtz, and T. A. Steiner. Differentially private substring and document counting. *Proc. ACM Manag. Data*, 3(2):95:1–95:27, 2025.

[6] P. Bille, A. R. Christiansen, P. H. Cording, I. L. Gørtz, F. R. Skjoldjensen, H. W. Vildhøj, and S. Vind. Dynamic relative compression, dynamic partial sums, and substring concatenation. *Algorithmica*, 80(11):3207–3224, 2018.

[7] L. Bonomi and L. Xiong. A two-phase algorithm for mining sequential patterns with differential privacy. In *Proc. 22nd CIKM*, pages 269–278, 2013.

[8] M. Bun, C. Dwork, G. N. Rothblum, and T. Steinke. Composable and versatile privacy via truncated CDP. In *Proc. 50th STOC*, pages 74–86, 2018.

[9] M. Bun, K. Nissim, U. Stemmer, and S. P. Vadhan. Differentially private release and learning of threshold functions. In *Proc. 56th FOCS*, pages 634–649, 2015.

[10] R. Chen, G. Ács, and C. Castelluccia. Differentially private sequential data publication via variable-length n-grams. In *Proc. 19th CCS*, pages 638–649, 2012.

[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. 18th KDD*, pages 213–221, 2012.

[12] E. Cohen, X. Lyu, J. Nelson, T. Sarlós, and U. Stemmer. Optimal differentially private learning of thresholds and quasi-concave optimization. In *Proc. 55th STOC*, pages 472–482, 2023.

[13] C. Dwork and J. Lei. Differential privacy and robust statistics. In *Proc. 41st STOC*, pages 371–380, 2009.

[14] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. 3rd TCC*, volume 3876, pages 265–284. Springer, 2006.

[15] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proc. 42nd STOC*, pages 715–724, 2010.

[16] M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. 38th FOCS*, pages 137–143. IEEE Computer Society, 1997.

[17] M. Farach-Colton, P. Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000.

[18] Y. Gao. Adaptive data structures for 2d dominance colored range counting. In *Proc. 18th WADS*, pages 460–473, 2023.

[19] Y. Gao and M. He. Space efficient two-dimensional orthogonal colored range counting. In *Proc. 29th ESA*, pages 46:1–46:17, 2021.

[20] B. Ghazi, P. Kamath, R. Kumar, P. Manurangsi, and K. Wu. On differentially private counting on trees. In *Proc. 50th ICALP*, volume 261, pages 66:1–66:18, 2023.

[21] B. Ghazi, R. Kumar, J. Nelson, and P. Manurangsi. Private counting of distinct and k-occurring items in time windows. In *Proc. 14th ITCS*, pages 55:1–55:24, 2023.

[22] H. Kaplan, K. Ligett, Y. Mansour, M. Naor, and U. Stemmer. Privately learning thresholds: Closing the exponential gap. In *Proc. 33rd COLT*, pages 2263–2285, 2020.

[23] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38(3):982–1011, 2008.

[24] T. Khatri, G. G. Dagher, and Y. Hou. Privacy-preserving genomic data publishing via differentially-private suffix tree. In *Proc. 15th EAI*, pages 569–584, 2019.

[25] K. Kim, S. Gopi, J. Kulkarni, and S. Yekhanin. Differentially private n-gram extraction. In *Proc. 34th NeurIPS*, pages 5102–5111, 2021.

[26] G. M. Landau, T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. 12th CPM*, pages 181–192, 2001.

[27] M. Nunez-del Prado, Y. Maehara-Aliaga, J. Salas, H. Alatrista-Salas, and D. Megías. A graph-based differentially private algorithm for mining frequent sequential patterns. *Applied Sciences*, 12(4), 2022.

[28] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[29] J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *Proc. SIGMOD Conference 2016*, pages 155–170, 2016.