Privacy-Preserving Genomic Data Publishing via Differentially-Private Suffix Tree

Tanya Khatri, Gaby G. Dagher *, and Yantian Hou

Department of Computer Science
Boise State University

Abstract. There is a huge need for sharing genomic data to advance medical and health researches. However, since genomic data is highly sensitive and the ultimate identifier, it is a big challenge to publish genomic data while protecting the privacy of individuals in the data. In this paper, we address the aforementioned challenge by presenting an approach for privacy-preserving genomic data publishing via differentially-private suffix tree. The proposed algorithm uses a top-down approach and utilizes the Laplace mechanism to divide the raw genomic data into disjoint partitions, and then normalize the partitioning structure to ensure consistency and maintain utility. The output of our algorithm is a differentially-private suffix tree, a data structure most suitable for efficient search on genomic data. We experiment on real-life genomic data obtained from the Human Genome Privacy Challenge project, and we show that our approach is efficient, scalable, and achieves high utility with respect to genomic sequence matching count queries.

Keywords: Genomic Data, Differential Privacy, Privacy-Preserving Data Publishing

1 Introduction

In this revolutionary era of science and technology, it is much easier to access, analyze and interpret genomic data. Genomes are the most vital part of the human, as they include health and other information about the person as well as their ancestors, siblings and decedents. Since the sharing of genomic data is essential for the advancement of genomic research, it is important to ensure that sensitive information about individuals in their genomic sequences are protected in any shared data for scientific research. Maintaining the correct trade-off between utility and privacy is particularly challenging for genomic data as each individuals' DNA sequence is unique and therefore, a DNA sample can never be made truly anonymized.

Health Insurance Portability and Accountability Act (HIPAA) [2] and Genetic Information Nondiscrimination Act (GINA) [3] are frameworks provided by the government to achieve the aforementioned delicate balance. Genome Wide Association Studies (GWAS) investigate genomic and biometric data with the purpose of identifying genetic variations that may be linked to diseases. The goal of GWAS is to produce aggregate statistics that are produced by examining many single nucleotide polymorphism

* corresponding author

Table 1: Homer's Attack : The attacker knows the genome of the victim (set of variants), the size of the mixture he's attacking and the population allele frequencies. [17]

Id	Allele Frequency			Distance Measure	Inference
	Reference Population(p)	Mixture(m)	Person of Interest(x)	$D(x) = \|x-p\| - \|x-m\|$	
j	0.25	0.75	1.0	0.50	Most likely to be in the Mixture
j+1	0.25	0.75	0.50	0.00	Equally likely to be in the Mixture and in the Reference Population
j+2	0.25	0.75	0.0	-0.50	Most likely to be in the Reference Population

(SNPs) locations from a group of study participants. This can be achieved by calculating *chi-squared* and *p-value* statistics. Since the aggregate statistics are taken from a sample of thousands of individuals, researchers believed that privacy was preserved by using de-identification techniques and a large sample size.

With the emergence of cloud computing, the possibility of large scale distribution of data collection from multiple resources has increased, as has the threat to privacy or information leakage. Recent work has shown, however, that the large volume of data collected from each patient exposes them to privacy breaches, even if only the aggregate statistics are reported. Homer *et al.* [17] show that a participant's information can be inferred from the allele frequencies of a large number of single-nucleotide polymorphisms (SNPs). Given the minor allele frequencies (MAFs) of both a reference population and a test population, the presence of an individual with a known genotype can be inferred using a t-test and a distance metric designed to contrast similarity between an individual and the test population. Table 1 shows an example statistics for Homer's attack, where given the genome of the victim (set of variants), the size of the mixture and the population allele frequencies, an attacker can re-identify an individual in a case group with a certain disease. by calculating the distance measure $D(x) = |x-p|-|x-m|$. If $D > 0$, it means that an individual is most likely to be in the mixture and if $D < 0$, it means that an individual is most likely to be in the reference population. $D = 0$ means equally likely to be in the mixture and in the reference population. Wang *et al.* [28] show that a participants' actual genomes can be reconstructed using correlation information about the SNPs. There are many other attacks [15][14][24] that could result in breaching the privacy of individuals.

In this paper, we address the problem of privacy-preserving genomic data publishing by proposing an approach that efficiently perturbs the raw genomic data and outputs an anonymized suffix tree that is privacy-preserving to effectively and efficiently support count queries for genomic sequence matching. To generate a privacy-preserving suffix tree, we utilize differential privacy [8], a rigorous privacy model that provides strong privacy guarantees independent of an adversary's background knowledge and computational power. Differential privacy is typically achieved through random perturbation, noise is carefully calibrated to the sensitivity. A differentially-private mechanism ensures that all outputs are insensitive to any individual's data. In other words, an individual's privacy is not at risk because of her participation in the dataset. Figure 1 illustrates a ϵ-differentially-private version of the raw data. An attacker can not infer anything about an individual by looking at ϵ-differentially-private data, regardless of any background knowledge about an individual.

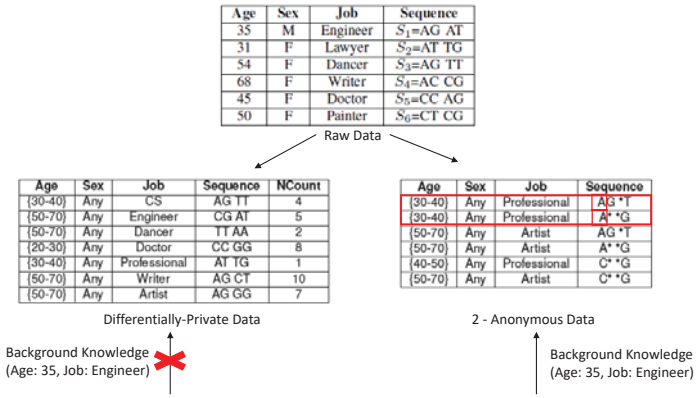

Fig. 1: Linkage Attack

1.1 Contributions

The contributions of this paper can be summarized as follows:

- We propose a novel non-interactive approach for anonymizing genomic data and releasing differentially-private suffix tree with an effective utility for genomic sequence matching. We are the first to publish differentially-private suffix tree for efficient searching on genomic data.
- We propose a top-down partitioning approach using Laplace mechanism to efficiently process datasets containing large number of genomic sequences.
- To obtain higher utility, we apply a normalization technique on the partitions to ensure consistency among genomic sequences and their suffixes.
- We implemented our approach and performed extensive experiments on real-life genomic data obtained from Human Genome Privacy Challenge [1]. The results show that our approach is efficient, scalable, and achieves high utility with respect to count queries.

2 Problem Overview

Let $\mathcal{D}$ be a raw genomic data containing n sequences $\{S_1, \ldots, S_n\}$, where each sequence consists of a number of SNPs, for example, $S_1 = AGAT$. To achieve differential privacy, the principal approach is to perturb the true output by adding a random noise to it. To generate the noise according to Laplace distribution, we need to determine the privacy budget B beforehand. The suffix tree is one of the most important and widely used data structures in bio-informatics and comparative genomics. It is a special data structure, with a wide range of application, including exact matching problems, substring problems, data compression and circular strings. Suffix tree is crucially important in sequencing and investigating DNA, such as looking for the longest common substring of two DNA sequences, finding exact and inexact matchings of a sample in a long sequence. The motivation of this work is that in computational biology, molecular biology and bio informatics these problems are crucially important. In our approach, we aim to generate a differentially-private suffix tree $\mathcal{T}$ that will support count queries. Count queries, as a general data analysis task, are the building block of many data mining tasks. We denote by *user count query* any data mining's count query that attempts to

answer the following question: how many times a specific pattern u occurs in the data, i.e., suffix-tree $\mathcal{T}$?

3 Proposed Solution

3.1 Differentially-Private Genomic Data Publishing

This main algorithm (Algorithm 1.1) takes as input a raw genomic data $\mathcal{D}$, a privacy budget B and user-specified height *h*, and returns a suffix tree $\mathcal{T}$ satisfying B-differential privacy, as shown in Algorithm 1.1. In Step 1, it calls Algorithm 1.2 to construct a noisy partitioning tree $\mathcal{P}$ based on Laplace noise and specialization threshold θ. In Step 2, Algorithm 1.3 is called, which takes as input a differentially-private partitioning tree $\mathcal{P}$, and normalize $\mathcal{P}$ based on the utility constraints to maintain the usefulness of the outputted normalized partitioning tree $\hat{\mathcal{P}}$. In Step 3, Algorithm 1.4 is executed to construct a differentially-private suffix tree $\mathcal{T}$ based on the differentially-private suffixes obtained from the normalized partitioning tree $\hat{\mathcal{P}}$. In Step 4, the algorithm outputs a differentially-private suffix tree $\mathcal{T}$.

Differentially-Private Genomic Data Publishing Algorithm
Input: Genomic Data $\mathcal{D} = \{S_1, \ldots, S_n\}$, Privacy Budget B, Height of the tree h
Output: Differentially-private suffix tree $\mathcal{T}$

1. Execute Algorithm 1.2 to construct a partitioning tree $\mathcal{P}$ based on $\mathcal{D}$ and B.
2. Execute Algorithm 1.3 to normalize $\mathcal{P}$ according to the utility constraint 3.3
3. Execute Algorithm 1.4 to construct a differentially-private suffix tree $\mathcal{T}$ from $\hat{\mathcal{P}}$.
4. Return suffix tree $\mathcal{T}$.

Algorithm 1.1: Differentially-Private Genomic Data Publishing

3.2 Partitioning Tree Generation

In this phase, our approach is to generate differentially-private suffixes, to ensure privacy-preserving data publishing. Our strategy for generating suffixes in a differentially-private manner is to use a top-down approach based on constructing disjoint partitions for multiple levels using Laplace mechanism and specialization threshold θ. Therefore, we construct a partitioning tree $\mathcal{P}$ by recursively grouping sequences in $\mathcal{D}$ into disjoint sub-datasets based on their suffixes. Given a raw genomic data $\mathcal{D} = \{S_1, \ldots, S_n\}$, privacy budget B and user specified height *h*, Algorithm 1.2 generates a differentially-private partitioning tree $\mathcal{P}$. Each partition contains four values: the nucleotide type, the suffix, sequences containing the suffix and the noisy count associated with it as described in Definition 3.2. In Step 1, we compute the specialization threshold $\theta = c * 2\sqrt{2k/B_l}$ (two times the standard deviation of noise) [7], where *c* is a constant that will be determined through experiments, *k* is the length of the sequence and B_l is the privacy budget per level.

Step 2 creates a virtual root partition *r*, where a partition is a data structure formally defined as follows.

Partition. *A partition is a tuple with four values [*Nuc, UNuc, Seqs, NCount*], where:*

- *Nucleotide (*Nuc*) is a type of bases adenine, guanine, thymine, and cytosine* A, G, T, C *in a strand of DNA.*

Differentially-Private Partitioning Tree Generation Algorithm
Input: Genomic Data $\mathcal{D} = \{S_1, \ldots, S_n\}$, Privacy Budget B, Height of the tree h
Output: Differentially-private partitioning tree $\mathcal{P}$

1. Compute the specialization threshold $\theta = c * 2\sqrt{2k/B_l}$, where k is the length of the sequence.
2. Construct a partitioning tree $\mathcal{P}$ with a virtual root partition *r*:
   (a) Assign all sequences in $\mathcal{D}$ to r: $r.Seqs = \{S_1, \ldots, S_n\}$
   (b) Set nucleotide $r.Nuc$ to *Any*.
   (c) Set $r.UNuc$ and noisy count $r.NCount$ to *Null*.
3. For each nucleotide type A, C, G and T, create a child partition v:
   (a) Set $v.Nuc$ to the nucleotide type.
   (b) Set $v.UNuc$ to $v.Nuc \cup Parent(v).UNuc$.
   (c) For each sequence S in $Parent(v).Seqs$, assign S to $v.Seqs$ *iff* $v.UNuc$ is a suffix of S.
   (d) Generate a Laplace noise $L_{noise} = Lap(2 \times k \times h/(B_l - \sqrt{B_l}))$.
   (e) Set $v.NCount$ to $|v.Seqs| + L_{noise}$, where $|v.Seqs|$ is the number of sequences assigned to partition v.
4. $h \leftarrow h - 1$.
5. While $h > 0$, for each child partition v created in Step 3, if $v.NCount \geq \theta$, then repeat Steps 3 and 4.
6. Return $\mathcal{P}$.

Algorithm 1.2: Differentially-Private Partitioning Tree Generation

- UNuc *keeps track of the suffixes of* Seqs = $\{S_1, \ldots, S_n\}$ *for each child partition.*
- Seqs = $\{S_1, \ldots, S_n\}$ *is the set of sequences that ends with the suffix* UNuc.
- NCount *is the noisy count which is the summation of count of sequences* Seqs *and the Laplace noise.* □

All sequences in $\mathcal{D}$ are initially assigned to $r.Seqs$, $r.UNuc$ is set to *Any*, and *r.UNuc* and *r.NCount* are set to *Null*. In Step 3, for each nucleotide type *A, G, T* and *C*, we create child partition v for r. For each child partition v, we assign $v.Nuc$ as the nucleotide type and $v.UNuc$ as $v.Nuc \cup Parent(v).UNuc$. For each sequence we determine if $v.UNuc$ is a suffix of S, then assign it to $v.Seqs$. The variance of the Laplace noise is $2 * S(f)/(B_l - \sqrt{B_l})$, where $S(f)$ is the sensitivity and $(B_l - \sqrt{B_l})$ is the privacy budget used for partitioning. The noisy count associated with each partition is the sum of the true count and the Laplace noise. To build $\mathcal{P}$, we use uniform budget allocation scheme. We divide the total privacy budget B equally, i.e., the privacy budget used per level for constructing $\mathcal{P}$ is $B_l = B/h$. One important observation is that all partitions on the same level contain disjoint sequence sets, and therefore the privacy budget allocated to a level can be used in full for each partition in it. In Step 5, for each child partition v created in Step 3, we continue to create further child partitions if $v.Ncount \geq \theta$ for that partition and height $h > 0$. Step 6 outputs the differentially-private partitioning tree $\mathcal{P}$. Figure 2 provides an example of $\mathcal{P}$ from raw data in Figure 1.

Privacy Budget Allocation B-differential privacy can be achieved by applying a differentially-private mechanism, commonly Laplace mechanism [9], that consumes a privacy budget B and calibrates noise according to the global sensitivity $S(f)$ of a func-

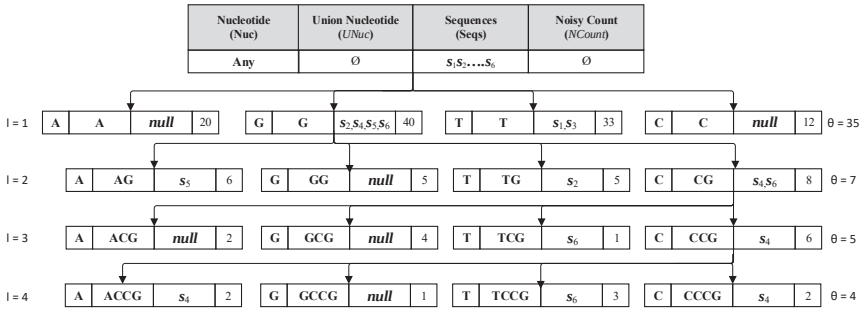

Fig. 2: Partitioning Tree

tion f. To address the existing utility and privacy trade-off, a geometric mechanism [12] was proposed which is a factor of an optimal mechanism $\mathcal{M}_u$ that is user-independent for every user u [6]. As in our approach we will be normalizing the generated partitioning tree $\mathcal{P}$ so that leaf nodes would be least noisy, therefore, to construct partitions at each level, we employ a uniform budget allocation scheme. That is, the privacy budget allocated to each level l is $B_l = B/h$, which is used for constructing partitions at level l in $\mathcal{P}$, as well as to compute the specialization threshold θ.

3.3 Bottom-Up Normalization

To ensure utility (usefulness) and consistency of the data, a bottom-up approach is proposed to normalize the partitioning tree based on the following utility constraint.

Utility Constraint. The noisy count $NCount$ of any node v in the partitioning tree $\mathcal{P}$ should be greater or equal to the total sum of its children's noisy counts. That is:

$\forall v \in \mathcal{P},\ v.NCount \geq \sum_{u \in child(v)} u.NCount$ □

Algorithm 1.3 takes the differentially-private partitioning tree $\mathcal{P}$ as an input and updates the noisy count $NCount$ of partitions from leaf to root to generate a normalized partitioning tree $\hat{\mathcal{P}}$. Following the bottom-up approach, we start with level $l = h - 1$, where h is the height of the differentially-private partitioning tree $\mathcal{P}$. The algorithm ensures that for each non-leaf node in $\mathcal{P}$, the utility constraint holds true.

Bottom-Up Normalization Algorithm
Input: Differentially-private partitioning tree $\mathcal{P}$
Output: Normalized partitioning tree $\hat{\mathcal{P}}$
Apply a bottom-up approach on $\mathcal{P}$ to obtain a normalize partitioning tree $\hat{\mathcal{P}}$:

1. $l \leftarrow h - 1$, where h is the height of $\mathcal{P}$.
2. For each non-leaf node v in $\mathcal{P}$ at level l, if the utility constraint 3.3 is not satisfied, then update the noisy count of v:
   $v.NCount \leftarrow \sum_{u \in child(v)} u.NCount$.
3. $l \leftarrow l - 1$.
4. While $l > 0$, repeat Steps 2 and 3.
5. Return $\hat{\mathcal{P}}$.

Algorithm 1.3: Bottom-Up Normalization

Suffix Tree Generation Algorithm
Input: Normalized partitioning tree $\hat{\mathcal{P}}$
Output: Differentially-private suffix tree $\mathcal{T}$
Apply a top-down approach on $\hat{\mathcal{P}}$ to obtain a differentially-private suffix tree $\mathcal{T}$:

1. Set the initial level: $l \leftarrow 1$.
2. For each node v in $\hat{\mathcal{P}}$ at level l:
   (a) Use $v.UNuc$ sequence to create a tree branch in $\mathcal{T}$.
   (b) Append $v.NCount$ dollar signs \$ to the added branch.
3. $l \leftarrow l + 1$.
4. While $l \leq h$, where h is the height of $\hat{\mathcal{P}}$, then repeat Steps 2 and 3.
5. Return $\mathcal{T}$.

Algorithm 1.4: Suffix Tree Generation

If the utility constraint 3.3 is not satisfied, that is, the noisy count of the parent is less than the sum of the noisy count of its children, then the parent's noisy count is updated to be the sum of the noisy count of all its children, as described in Step 2. We keep normalizing based on the utility constraint until we reach the root partition of the tree. In Step 5, a differentially-private normalized partitioning tree $\hat{\mathcal{P}}$ is produced as an output.

3.4 Suffix Tree Generation

This algorithm (Algorithm 1.4) takes as an input the normalized partitioning tree $\hat{\mathcal{P}}$, and based on a top-down approach it outputs the B-differentially-private suffix tree $\mathcal{T}$. In the normalized partitioning tree, $U.Nuc$ represents suffix and $NCount$ represents the normalized noise count. We traverse the normalized partitioning tree level-wise and create a branch in the suffix tree for each suffix present in $\hat{\mathcal{P}}$. Starting with level 1, for each partition v in a normalized tree $\hat{\mathcal{P}}$, we use the suffix presents in $v.UNuc$ and its corresponding $v.NCount$ to create a tree branch in the suffix tree $\mathcal{T}$, as shown in Figure 3, and add the associated noisy count at the end. For each added branch, we append the dollar sign \$ equal to the noisy count $v.NCount$. Every path from the root to the square node represents a suffix. Because the tree is normalized, there is no violation of utility constraint when we take the parent suffix and add it to the suffix tree i.e., following a top-down approach. Since, the noisy count of the parent partition is always grater than or equal to the noisy count of its children, the suffixes maintain consistency. Step 5 outputs the differentially-private suffix tree $\mathcal{T}$ that support count queries for genomic sequence matching and maintains data utility.

4 Algorithm Analysis

4.1 Privacy Analysis

We use the composition properties of differential privacy to guarantee that the proposed algorithm satisfies B-differential privacy as a whole. Any sequence of computations that each provides differential privacy in isolation also provides differential privacy in sequence, which is known as *sequential composition*.

Sequential composition [22]. Let each A_i provide B_i-differential privacy. A sequence of $A_i(\mathcal{D})$ over the dataset $\mathcal{D}$ provides $(\sum_i B_i)$-differential privacy. □

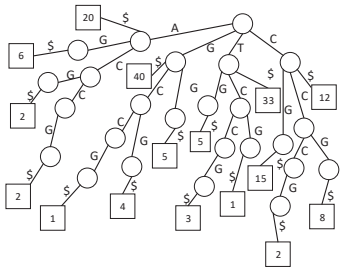

Fig. 3: Differentially-Private Suffix Tree generated from the normalized version of partitioning tree $\hat{\mathcal{P}}$ shown in Figure 2.

However, if the sequence of computations is conducted on disjoint datasets, the privacy cost does not accumulate but depends only on the worst guarantee of all compositions. This is known as *parallel composition*.

Parallel composition [22]. Let each A_i provide B_i-differential privacy. A sequence of $A_i(\mathcal{D})$ over a set of disjoint datasets $\mathcal{D}$ provides B_i-differential privacy. □

Given a privacy budget B, our algorithm generates B-differentially-private genomic suffix tree. *Proof:* Our proposed algorithm consists of three phases - *partitioning tree generation*, *bottom-up normalization* and *suffix tree generation*. We have a fanout f = 4. According to Lemma 4.1, we can use the same privacy budget for each partition at the same level. Therefore, if we can prove that the summation of the privacy budget used in each partitioning level is less than or equal to B, we get the conclusion that our approach satisfies B-differential privacy.

In the partitioning tree generation step of our algorithm, the allocation of privacy budget per level is $B_l = B/h$, where h is the height of the partitioning tree $\mathcal{P}$. The privacy budget needed for computing the specialization threshold θ is $\sqrt{B_l}$. As all the partitions on the same level in the partitioning contain disjoint sets of sequences, according to Lemma 4.1, the total privacy budget for each partition to build the noisy partitioning tree $\mathcal{P}$ is $B_l - \sqrt{B_l}$. The only time we refer to the original data is when we compute total count of the sequences in each partition and as we mention above for each partition we compute $B_l - \sqrt{B_l}$ privacy budget. For the bottom-up normalization and suffix tree generation phase, no privacy budget is consumed as we are not utilizing the original data. Therefore, the total privacy budget consumed can be formulated as:

$$\sum_{l=1}^{h} \underbrace{(B_l - \sqrt{B_l})}_{\text{partition}} + \underbrace{\sqrt{B_l}}_{\text{threshold}} = h \times B_l \leq B$$

As proven by Hay *et al.* [16], a post-processing of differentially-private results remains differential private. Therefore, our approach satisfies B-differential privacy. □

4.2 Complexity Analysis

(Complexity). The overall complexity of our proposed approach in the average case is $\mathcal{O}(h^2 \times n)$. *Proof.* We can determine the time complexity of the proposed approach in terms of three phases: Partitioning tree generation, Normalization and Suffix tree generation.

Partitioning tree generation phase. In the partitioning tree generation phase, the runtime complexity is the cost of generating each partition times the number of partition in the tree. For each level l of the partitioning tree, the maximum number of possible partitions in the worst case is 4^l, and the total number is $\sum_{l=1}^{h} 4^l$. However, since threshold θ is utilized, that restricts the exponential growth of the partitions, as θ fluctuates according to the Laplace noise distribution, which balances the number of partitions [10]. Therefore, in the average case, the number of partitions per level is $P_l \approx 4 \times l \ll 4^l$, and the total number of partitions is:

$$\sum_{l=1}^{h} 4 \times l = 2 \times h^2$$

Given that we process n sequences per level, the total cost of constructing a partitioning tree in an average case is $\mathcal{O}(n \times h^2)$, where, n is the number of sequences and h is the height of the partitioning tree.

Normalization phase. In the normalization phase, we traverse the partitioning tree once. Therefore, the total cost to construct a normalized partitioning tree in an average case is: $\sum_{l=1}^{h} 4 \times l = 2 \times h^2 = \mathcal{O}(h^2)$.

Suffix tree generation phase. In the suffix tree generation phase, the runtime complexity is the time taken to insert a full sequence times the number of sequences generated. As stated by [27][30][13], a string of length h can be inserted in $\mathcal{O}(h)$. In the average case, the number of full sequences to be generated from the partitioning tree is $\mathcal{O}(h)$, where these full sequences will account for all the trimmed sequences (short inbuilt suffixes). Therefore, the average computational cost to build a suffix tree is $\mathcal{O}(h \times h) = \mathcal{O}(h^2)$.

Therefore, the overall complexity of our proposed approach in the average case is: $\mathcal{O}(h^2 \times n + h^2 + h^2) = \mathcal{O}(h^2 \times n)$. □

Given that $h \ll n$, we show in our experiment that our algorithm scales linearly w.r.t. a linear increase of n.

5 Performance Evaluation

5.1 Implementation Setup & Dataset

We implemented our algorithm in Java, and our experiments were conducted on a machine equipped with an Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz processor and 32.0 GB RAM, running Windows 10 64-bit operating system.

The humane SNPs were obtained from the Human Genome Privacy Challenge [1]. The chr2 and chr10 datasets contain 311 SNPs and 610 SNPs respectively. The length of the sequences is 200. The information of datasets is provided in Table 2.

Table 2: **Properties of Experimental Datasets**

Dataset	# of SNP Sequences	Length of Sequence
chr2	311 (29504091-30044866)	200
chr10	610 (55127312-56292137)	200

5.2 Experimental Results

In this section, our objective is to study the impact of enforcing differential privacy on the data quality in terms of determining optimal value of constant c, scalability, efficiency and utility.

Determining Optimal Value of Constant c Recall that $\theta = c * 2\sqrt{2k/B_l}$. To determine the optimal value of constant c for the specialization threshold θ, we randomly generate queries of varying length from the suffix tree and experiment data utility for different c values. We started experimenting with a $0.25 < c$, and noticed that θ was too big and the condition to partition further was never satisfied. Therefore, we experimented on c values from the range [0, 0.25]. As shown in Figure 4, the best results in terms of average relative error is for $c = 0.15$.

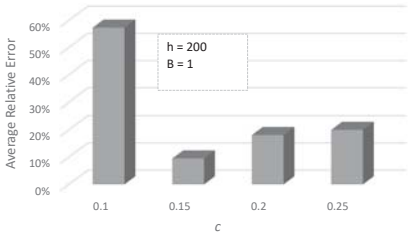

Fig. 4: Determining the optimal value of c for threshold θ

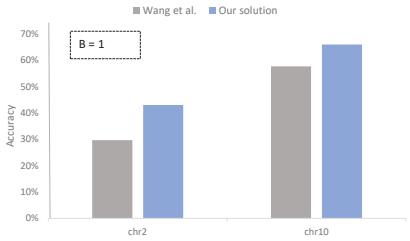

Fig. 5: Comparative evaluation of our proposed algorithm with [29]

Scalability The objective is to measure the runtime of each construction phase to ensure its capability to scale up in terms of record size. We measure the runtime of our proposed algorithm with respect to the linear increase in the number of sequences. We set the privacy budget B to 1, height of the tree to 200 and the value of c as 0.15. Figure 6 illustrates the runtime of our algorithm w.r.t. a linear increase in the number of sequences in the randomized dataset (200k, 400k, 600k, 800k and 1000k records). The x-axis represents the number of sequences and the y-axis shows the runtime in minutes. We observe that the growth in total runtime is linear when the data records increase linearly. We also observe that the partitioning tree construction phase is the most dominant phase in the algorithm, but they all scale linearly w.r.t. the number of sequences.

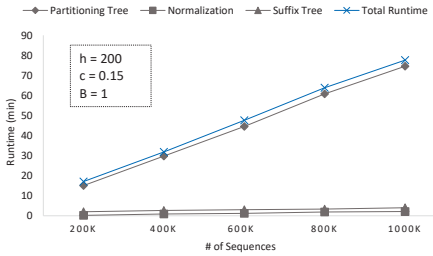

Fig. 6: Scalability w.r.t # of sequences

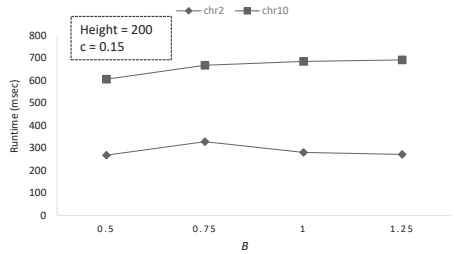

Fig. 7: Efficiency w.r.t budget B

Efficiency We evaluate the efficiency of our proposed approach on the chr2 and chr10 dataset with respect to the privacy budget B varying from 0.5 to 1.25 at an interval of 0.25. The x-axis represents privacy budget B and the y-axis represents the runtime

in milliseconds as shown in Figure 7. We observe that for chr2 dataset the runtime is between 200-300 milliseconds and for chr10 dataset the runtime is between 600-700 milliseconds. However, the runtime is consistent with the increase in B for both datasets.

Utility We measure the utility of a count query Q over the sanitized dataset $\tilde{\mathcal{D}}$ by its *relative error* with respect to the actual result over the raw dataset $\mathcal{D}$. The relative error is computed as:

$$\textit{Relative Error} = \frac{|Q(\tilde{\mathcal{D}}) - Q(\mathcal{D})|}{max(Q(\mathcal{D}), s)} \tag{1}$$

, where *s* is a sanity bound that is set to 1.

We randomly generate 500 counting queries with varying length of sequences. We divide the query set into five subsets such that the query length of the i-th subset is uniformly distributed in $[1, \frac{i}{5}\sqrt{|S|}]$ and each query is drawn randomly, where $|S| = 200$ is the length of the sequence . All relative error will be computed based on the average of 10 runs. The experimental results are generated using both datasets: chr2 and chr10. With 500 random queries of length $l \in [1, 14]$, since $i = 5$ in our experiments. The query length of the first subset for 20% of query length will be $[1, \frac{1}{5}\sqrt{200}]$, which is equal to [1,3]. Similarly,the five uniformly distributed subsets for 20%, 40%, 60%, 80% and 100% of query length are $[1, 3]$, $[1, 6]$, $[1, 8]$, $[1, 11]$, $[1, 14]$ respectively. We select five random queries for each sampling and the relative error is the average of 10 runs.

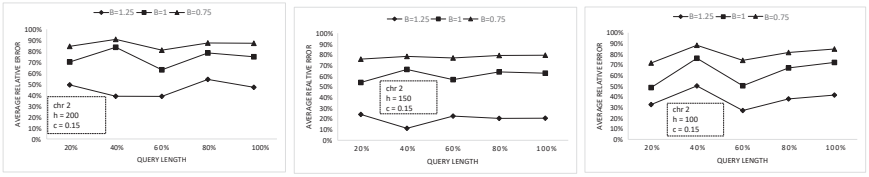

Fig. 8: Average relative error with respect to query length percentage for chr2 dataset

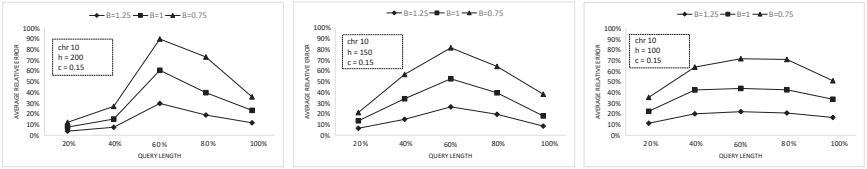

Fig. 9: Average relative error with respect to query length percentage for chr10 dataset

Figures 8 and 9 depict the average relative error as the value of privacy budget B grows from 0.75 to 1.25 at an interval 0.25, and height is set to 100, 150 and 200 with $c = 0.15$. The x-axis represent the maximum query length of each subset in terms of the percentage of $\sqrt{|S|}$ and the y-axis represent the average relative error. We observe that the average relative error decreases with the increase of B with respect to both datasets, and the best data utility is obtained for privacy budget $B = 1.25$. This observation

remains true irrespective of the height h of the tree, as shown in both figures for $h \in \{100, 150, 200\}$. This is due to the fact that as we increase the privacy budget, we are decreasing the noise that is being added and hence the data utility will be more. Also, we observe that in chr2 dataset (the smaller dataset), the average relative error is almost constant for each B and h, regardless of the query length. On the other hand, in chr10 dataset, the average relative error is variable depending on the query length. This is due to the fact in a larger dataset, the size of the data grows, and more noise is added to ensure privacy since the noisy count of more partitions will surpass threshold θ.

We also compare the performance of our proposed algorithm with the research work done by Wang *et al.* [29] for the differentially-private genome data dissemination through top-down specialization. We use the same dataset as in [29], and make the experimental settings as close as possible. We set the privacy budget $B = 1$, and determine the accuracy of our algorithm as follows:

$$\textit{Accuracy} = 1 - \textit{Average Relative Error} \tag{2}$$

, where the relative error is computed using Equation 1. Figure 5 illustrates that for both datasets chr2 and ch10, our proposed algorithm provides higher accuracy than [29]. That is, for $B = 1$, both algorithms achieves better accuracy on larger dataset (chr10), while our solution achieves higher accuracy with up to 70%.

6 Related Work

Genome-Wide Association Studies (GWAS) is used for analysis of sets of DNA sequences to discover and identify the genetic basis of disease. Although these statistics that are published as the result of GWAS can possibly be used for identification of the participating individuals. This has led to research on publishing GWAS data in a differentially-private manner. The papers [4][31][29][23][26][11][18] list a number of mechanisms to achieve differential privacy on genomic data. Akgün *et al.* [4] have categorized previously identified problems and their respective solutions introduced in research prior to theirs. Some of the problems and their solution techniques discussed by [4] are discussed in the following paragraphs in this section. Wang *et al.* [29] introduce an approach to disseminate differentially-private genomic data using top-down specialization. This method assumes a data owner has a data table $\mathcal{D}(A^i, A^{snp})$ where $\mathcal{A}^i$ are explicit identifiers and $\mathcal{A}^{snp}$ a set of SNPs. The algorithm aims to satisfy ϵ-differential privacy while retaining data utility with a high sensitivity and generates an anonymized data table $\hat{D}$ that can be released to the public. Uhler *et al.* [26] introduce methods that focus on releasing differentially-private minor allele frequencies, p-values, and χ^2 statistics for the M most relevant SNPs regardless of arbitrary external information. They also apply penalized logical regression technique while maintaining differential privacy guarantee to locate genome-wide associations in the data. Finally, for testing the approach, the proposed techniques are compared both on simulation data and on real canine hair length genomic data. This approach is an adaptation of Bhaskar *et al.* [5] to genome wide association studies. Yu *et al.* [31] extend the work of Uhler *et al.* [26] by allowing for an arbitrary number of cases and controls and performance of a risk-utility analysis. Next, the methods proposed are compared to the differentially-private publishing mechanism proposed by Johnson and Shmatikov [20]. The work of

Li *et al.* [21] introduces the compressive mechanism which uses a probabilistic compression procedure that generates a synopsis, adds Laplace noise to the compressed data, and then decodes the results. Compared to other synopsis proposals, the compressive sensing mechanism provides more accurate statistical query results while using less noise under certain conditions. Building upon this work, Roozgard *et al.* [25] presents a compressed sensing based, differentially-private genomic data dissemination algorithm that takes sequences of genomic nucleotides from multiple subjects, and transforms the frequencies of SNPs into a sparse vector representation. Laplace noise is then added to all elements of the sparse vector. Jiang *et al.* [19] suggests a method for releasing the top-K most significant SNPs across the genome when K is small.

7 Conclusion

When publishing genomic data, achieving the right balance between privacy and utility is challenging since each person's DNA sequence is unique (with the exception of identical twins) and a DNA sample therefore can never be made truly anonymized. In this paper, we propose an algorithm for privacy-preserving genomic data publishing via differentially-private suffix tree that is scalable, efficient and support count queries. We perform the privacy and complexity analysis of our approach and show that our approach preserves privacy and is scalable even with respect to large datasets. We also evaluate the performance of our algorithm and present the experimental to determine the scalability, efficiency and utility of our solution. The experiments demonstrated that our approach is scalable, efficient and maintains high utility to answer count queries.

Acknowledgement

This research was partially supported by Forsta, Inc (www.forsta.io).

References

1. Human genome privacy protection challenge.
2. Health insurance portability and accountability act (hipaa), 1996.
3. Genetic information nondiscrimination act (gena), 2008.
4. Mete Akgün, A Osman Bayrak, Bugra Ozer, and M Şamil Sağıroğlu. Privacy preserving processing of genomic data: A survey. *Journal of biomedical informatics*, pages 103–111, 2015.
5. Raghav Bhaskar, Srivatsan Laxman, Adam Smith, and Abhradeep Thakurta. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 503–512. ACM, 2010.
6. Luca Bonomi and Li Xiong. A two-phase algorithm for mining sequential patterns with differential privacy. In *Proceedings of the 22Nd ACM CIKM*, pages 269–278, 2013.
7. Rui Chen, Benjamin C.M. Fung, Bipin C. Desai, and Nériah M. Sossou. Differentially private transit data publication: A case study on the montreal transportation system. In *Proceedings of the 18th ACM SIGKDD on KDD*, pages 213–221, 2012.
8. Cynthia Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006.
9. Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.
10. Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, pages 211–407, 2014.

11. Stephen E Fienberg, Aleksandra Slavkovic, and Caroline Uhler. Privacy preserving gwas data sharing. In *IEEE International Conference on Data Mining Workshops*, pages 628–635, 2011.
12. Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.
13. R. Giegerich and S. Kurtz. From ukkonen to mccreight and weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, pages 331–353, 1997.
14. Michael T. Goodrich. The mastermind attack on genomic data. 2009.
15. Melissa Gymrek, Amy L. McGuire, David Golan, Eran Halperin, and Yaniv Erlich. Identifying personal genomes by surname inference. *Science*, 339:321–324, 2013.
16. Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3:1021–1032, 2010.
17. Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V. Pearson, Dietrich A. Stephan, Stanley F. Nelson, and David W. Craig. Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays. 2006.
18. Mr Zhicong Huang. Privacy preserving algorithms for genomic data.
19. Xiaoqian Jiang, Yongan Zhao, Xiaofeng Wang, Bradley Malin, Shuang Wang, Lucila Ohno-Machado, and Haixu Tang. A community assessment of privacy preserving techniques for human genomes. *BMC medical informatics and decision making*, 14(Suppl 1):S1, 2014.
20. Aaron Johnson and Vitaly Shmatikov. Privacy-preserving data exploration in genome-wide association studies. In *Proceedings of ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1079–1087, 2013.
21. Yang D Li, Zhenjie Zhang, Marianne Winslett, and Yin Yang. Compressive mechanism: Utilizing sparse representation in differential privacy. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 177–182, 2011.
22. Frank D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the SIGMOD '09*, pages 19–30, 2009.
23. Muhammad Naveed, Erman Ayday, Ellen W. Clayton, Jacques Fellay, Carl A. Gunter, Jean-Pierre Hubaux, Bradley A. Malin, and Xiaofeng Wang. Privacy in the genomic era. *ACM Comput. Surv.*, pages 6:1–6:44, 2015.
24. Laura L. Rodriguez, Lisa D. Brooks, Judith H. Greenberg, and Eric D. Green. The complexities of genomic identifiability.
25. Aminmohammad Roozgard, Nafise Barzigar, Pramode K Verma, and Samuel Cheng. Genomic data privacy protection using compressed sensing. *Transactions on Data Privacy*, 9:1–13, 2016.
26. Caroline Uhlerop, Aleksandra Slavković, and Stephen E Fienberg. Privacy-preserving data sharing for genome-wide association studies. *The Journal of privacy and confidentiality*, 5(1):137, 2013.
27. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, pages 249–260, 1995.
28. Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. Learning your identity and disease from research papers: Information leaks in genome wide association study. 2009.
29. Shuang Wang, Noman Mohammed, and Rui Chen. Differentially private genome data dissemination through top-down specialization. *BMC medical informatics and decision making*, 14(1):S2, 2014.
30. Peter Weiner. Linear pattern matching algorithms. SWAT '73, pages 1–11, 1973.
31. Fei Yu, Stephen E Fienberg, Aleksandra B Slavković, and Caroline Uhler. Scalable privacy-preserving data sharing methodology for genome-wide association studies. *Journal of biomedical informatics*, 50:133–141, 2014.