# Differentially Private Event Histogram Publication on Sequences over Graphs

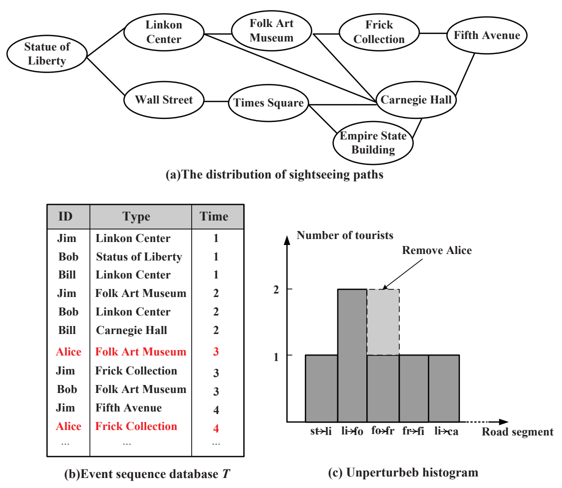
**Abstract** The big data era is coming with strong and ever-growing demands on analyzing personal information and footprints in the cyber world. To enable such analysis without privacy leak risk, differential privacy (DP) is quickly rising in recent years, as the first practical privacy protection model with rigorous theoretical guarantee. This paper discusses how to publish differentially private histograms on events in time series domain, with sequences of personal events over graphs with events as edges. Such individual-generated sequences commonly appear in formalized industrial workflows, online game logs and spatial-temporal trajectories, the direct publication of which may compromise personal privacy. While existing DP mechanisms mainly target at normalized domains with fixed and aligned dimensions, our problem raises new challenges when the sequences could follow arbitrary paths on the graph. To tackle the problem, we reformulate the problem with a three-step framework, which 1) carefully truncates the original sequences, trading off errors introduced by the truncation with those introduced by the noise added to guarantee privacy, 2) decomposes the event graph into *path sub-domain*s based on a group of event *pivots*, and 3) employs a deeply optimized tree-based histogram construction approach for each sub-domain to benefit with less noise addition. We present a careful analysis on our framework to support thorough optimizations over each step of the framework, and verify the huge improvements of our proposals over state-of-the-art solutions.

**Keywords** differential privacy, histogram publication, sequence, graph

## 1 Introduction

Huge amount of private and sensitive personal data are accumulating in the cyber world, along with the rapid IT technical advances such as workflow management systems [**?**, **?**], mobile Internet [**?**, **?**] and cookie-based browser tracking techniques [**?**]. The availability of such big time series data enables us to model the product logistics, analyze the visiting patterns of tourists and understand web surfing behaviors, any of which is crucial and worth billion dollars in the business. However, even simple aggregation over the time series domain may pose serious risks on personal privacy, as each time series in the cyber space contains sensitive information of individuals from the physical world.

Interestingly, many of such sequences generally follow a dependence relationship on a graph of concrete events. In Figure 1, we present an example on the analysis over sightseeing paths in *New York city*, in which each vertex in the graph in Figure 1(a) represents one popular sightseeing site, e.g. *Statue of Liberty*, and an edge denotes the tourist's movement from one site to another site. On the back end, a database system, e.g. the table in Figure 1(b), records the trajectories of tourists as site sequences, and is queried and analyzed to mine popular sightseeing tours and congestion trends. A commonly used tool for such analysis is event histogram, which publishes the numbers of tourists associated with all road segments, e.g. the bar chart in Figure 1(c) generated based on the database *T*.



| ID | Type | Time |
|---|---|---|
| Jim | Linkon Center | 1 |
| Bob | Status of Liberty | 1 |
| Bill | Linkon Center | 1 |
| Jim | Folk Art Museum | 2 |
| Bob | Linkon Center | 2 |
| Bill | Carnegie Hall | 2 |
| Alice | Folk Art Museum | 3 |
| Jim | Frick Collection | 3 |
| Bob | Folk Art Museum | 3 |
| Jim | Fifth Avenue | 4 |
| Alice | Frick Collection | 4 |
| ... | ... | ... |

Fig. 1. Example of event histogram on sightseeing trajectory database.

Unfortunately, direct publication of such histogram without perturbation is risky on privacy leakage. Assume that we remove the sequence of *Alice*'s trajectory from the database, which contains sightseeing sites *Folk Art Museum* and *Frick Collection*, the corresponding count over the edge from *Folk Art Museum* to *Frick Collection* decreases by one. Such deviation may inspire adversaries to infer *Alice*'s trajectory and furthermore her personal interests, if adversaries hold background knowledge on all trajectories except *Alice's*.

The techniques proposed in this paper target to support event histogram publication under differential privacy (DP) [**?**], to enable privacy-aware aggregation on counts of events along a specified path over the graph. Such aggregation is important and useful for decision makers to learn the density of particular paths over the graph. Different from traditional suppression techniques on sequence data [**?**, **?**, **?**], our proposals enhance the privacy protection with rigorous theoretical guarantee, as DP is robust against adversaries with strong background knowledge. To enforce DP, our proposals feature three new techniques to tackle the new challenges rising over the time series domain.

Firstly, the high sensitivity of sequences causes high noise scale and low utility of the perturbed histograms, as every sequence contributes to a number of counts in the histogram. The traditional approach, e.g. [**?**], transforms the original time series and retrieves a small number of significant Fourier coefficients, leading to much smaller global sensitivity and thus requiring much less noise for privacy protection. The accuracy of such approach relies on a careful selection on the coefficients after transformation, which is generally difficult in practice when the energy distribution in the frequency domain is highly dynamic. The truncation-based approach, on the other hand, simply cuts the original sequence to reduce the sensitivity [**?**, **?**]. However, existing solutions with truncation are mostly tricky on the selection of truncation length, due to the lack of theoretical analysis. In this paper, we fill the gap between theory and practice, by a new method to automatically choose a preferable truncation length for our sequence domain based on the privacy-aware statistics of the sequences.

Secondly, it is challenging to optimize for sequential aggregation because of the complexity on event graph structures. Traditional aggregation optimization methods for DP mainly focus on normalized domains, e.g. [**?**, **?**]. These methods are not directly applicable on our problem setting, as an event in the graph may be covered by totally different sequences. To reuse the theoretical results and experience learnt in previous studies, we propose to decompose the graph into paths and regard each path as a sub-domain. Such decomposition is non-trivial, as it is crucial to select appropriate *event pivot*s in the event graph as the boundary of the result sub-domains. By leveraging the distribution of queries, we design a *min-cut* strategy to minimize the number of sub-queries induced by the domain partitioning, thus improving the general accuracy of the query results.

Thirdly, the non-uniformity of aggregation query workload raises new needs of optimizations on the noise injection procedure. Traditional optimization strategies are commonly designed based on the uniform workload assumption, such that the count aggregation is uniformly chosen by the database user. This potentially leads to unexpectedly poor performance, as the users are more likely to query on more important events than the others. Our approach allows users to specify a group of workload sample queries, with which the system automatically constructs an optimal tree structure on sub-domains, in order to minimize the expected error in terms of the given workload. As a short summary, the major contributions of our work are summarized as below:

1. We design a three-step framework, including truncation length analysis, event graph decomposition and tree-based histogram construction.

2. To reduce the sensitivity, we learn a truncation length for original sequences based on differentially private sequence length statistics.

3. During decomposing the event graph into sub-domains, we propose a *min-cut* strategy to reduce the number of involved sub-queries induced by sub-domain partitioning, thus reducing the error of query results.

4. We propose a new optimization method for tree-based histogram construction of each sub-domain and leverage the query workload to automatically adjust the fanout of the tree.

The remainder of this paper is organized as follows. Section 2 reviews the existing studies on privacy preservation on time series data and state-of-the-art methods for DP. Section 3 presents preliminaries about DP and problem definition. Section 4 discusses how to truncate time series data for better querying performance. Section 5 designs a new graph decomposition strategy to transform the original event graph to sub-domains. Section 6 devises new optimization techniques on each sub-domain based on the given query workload. Section 7 evaluates the usefulness of our proposals, and Section 8 concludes the paper.

## 2 Related Work

Event sequence data often follow pre-defined dependence relationships in a variety of application domains, such as complex event processing [?, ?], spatial data management [?, ?] and business processes [?, ?]. Since the event sequence data record human behaviors, direct statistics publication may divulge the privacy of individuals. Privacy preservation on event sequence data is an important research topic in different areas of computer science. Existing studies on this topic can be basically categorized into three classes.

**Time-series publication under DP** [?, ?, ?, ?]: Given an event sequence pattern, these methods return the appearance count of the pattern in all sequences in the database. Chen et al. [?] propose to publish the counts, by an exploration tree structure to maintain the essential information of the sequential database. Besides, He et al. [?] apply hierarchical reference systems to publish the noisy trajectories composed of GPS points. Specially, they adopt grids of different sizes to discretize the GPS locations, then employ multiple differentially private exploration trees to organize the discrete GPS trajectories. To improve the query result accuracy, Chen et al. [?] and Zeng et al. [?] independently introduce truncation-based solutions over event sequence domain. But the truncation length is decided based on experiment evaluations or domain knowledge, with no theoretical analysis on the validity and optimality of their methods. And [?] utilizes the length of one sequence to normalize the counts that it contributes to. The normalization has an effect on reducing sensitivity, but introduces bias because of the different weights of sequences.

These studies all concentrate on the count of event sequence publication, with the queries defined to count the appearances of a particular event sequence instead of aggregation of counts on sequences along the paths in the graph. If applied on our problem setting, each query is evaluated by summing up a group of noisy counts. The accuracy based on their approaches generally drops quickly, when more noisy counts are involved.

**Event Count Publication** [?, ?]: These methods return counts for range queries. For event sequence data, a sequence from one individual affects the counts of multiple events. In most cases, such queries have to accept errors of large scale, because of the high sensitivity of the queries. To handle this problem, [?] employs Fourier Perturbation Algorithm (FPA) with two steps, including (i) derivation of a subset of its Discrete Fourier Transform (DFT) coefficients and (ii) coefficients perturbation. Such method generates smaller noise scale than the standard Laplace mechanism [?].

However, the accuracy of the results mostly relies on the quality of the DFT approximation. [?] proposes the GS method to improve the result accuracy by adding noises to averages of groups. Firstly, their method sorts the column attributes based on the counts of every column, followed by decomposition on the sorted column attributes into disjoint groups. The average is calculated as the representative value of every group, with final noise injection on each group average. However, GS incurs additional errors by regarding group average as the value of any column. Besides, for range queries, sorting decreases the possibility that one group covers the target range, leading to needs of overwhelming noises.

There are some other studies on counting query processing on normalized domain for range queries under DP. Hay et al. [?] present a method to decompose the data attribute interval by a hierarchical $k$-tree, in which every node denotes the frequency of an interval and has $k$ children corresponding to $k$ equal subintervals. Given the tree structure, Laplace noises are added to nodes in the tree. In addition to tree construction, their method also adjusts the values of the nodes for consistency, such that noisy count of the parent node is equal to the sum of its children nodes. Qardaji et al. [?] improve this method by finding optimal $k$ for the equal-tree structure. Basically, all these methods assume queries are generated from the range of the attribute and build models to estimate the effects of tree fanouts. Although these methods could support long range queries, there are clear limitations, including (i) the uniform-tree structure does not allow different fanouts at different nodes in the tree structure, potentially losing further optimization opportunities; and (ii) queries are under uniform distribution assumption on the normalized domain. Such limitations render suboptimal performance in our event histogram publication, because each normalized sub-domain has a different non-uniform workload.

**Linear Counting Query Processing**[?, ?, ?]: Our problem is also related to linear counting query processing under differential privacy, which aims to return aggregation on selected counts on a discrete domain without particular order. Similar to our problem, in linear counting query optimization, the database system is given a number of sample queries as the workload. Yuan et al. [?] and Li et al. [?, ?] propose approaches to utilize matrix-based optimizations, in order to minimize the expected error on a group of linear counting queries. Yuan et al. [?] use matrix mechanism to get the optimal strategy for workload, which requires solving high complexity optimization problems and has a bad scalability. And Li et al. [?, ?] improve the efficiency by giving the query strategies (hierarchical queries) in advance, then computing the the coefficients of given query strategies. However, linear counting queries simply assume each individual contributes to at most one count in the domain. The optimization is done on query workload, under this low sensitivity assumption. Besides, since the given strategies[?] are independent of a special workload, they cannot guarantee the accuracy for the workload even if optimal coefficients are computed for given strategies.

## 3 Preliminaries

### 3.1 Data and Query Model

We assume the sequential database $D$ consists of $n$ sequence records, i.e. $D = \{s_1, s_2, \ldots, s_n\}$. Each sequence record $s_i$ contains a series of $l_i$ discrete events, i.e. $s_i = (e_{i1}, e_{i2}, \ldots, e_{il_i})$, with each event $e_{ij}$ from the event set $E$. There exists a graph $G(V, E)$ defined over the event set, in which each vertex $v \in V$ connects events in $E$.

Given a sequence database $D$ and its associated graph $G$, we aim to publish an event histogram consisting of $|E|$ counts, i.e. $\{c_1, c_2, \ldots, c_{|E|}\}$ , where $c_k$ corresponds to the number of occurrences of event $e_k$, i.e. $c_k = |\{s_i \mid s_i \in D \wedge e_k \in s_i\}|$. The event histogram shows the count distribution of edges in the graph. Database users thereby can aggregate the counts along paths in $G$ to obtain statistics, called sequence range query. Each sequence range query $Q$ contains a sequence of events compatible with $G$, as defined below.

**Definition 1.** *An event sequence* $S = (e_1, e_2, \ldots, e_l)$ *is compatible with* $G(V, E)$ *if all edges in* $S$ *induce a path over* $G$.

Given a query $Q = (e_1, e_2, \ldots, e_l)$ and sequence database $D$, the result of $Q$ is calculated by aggregating the corresponding counts in the histogram, i.e. $Q(D) = \sum_{j=1}^{l} c_j$, where $c_j$ indicates the count of sequences in $D$ containing $c_j$. Such sequence range queries are useful and crucial for business intelligence analysis over sequential data. Next we list four applications as below.

**Traffic Violation** To perform the duties of safety promotion and collision reduction, the Traffic Bureau officers are supposed to investigate the distribution of hazardous violations by drivers on roads. For example, driver $A$ disobeys traffic rules in roads *shisan*-1, *shisan*-2 and *qingnian*-2. On the other hand, violations committed by driver $B$ also have been observed in roads *shisan*-2 and *nanwu*-1. Obviously, there exist 3 violations in road *shisan*, where one happens in road *shisan*-1 and the other two are conducted in road *shisan*-2. Both road *nanwu*-1 and road *qingnian*-2 have one violation. Note that the total number of violations in a road is the sum of ones in its sub-roads, rather than the number of persons who violate. Here, a violation is regarded as an event. Then, the result of sequence range query (*shisan*-1, *shisan*-2) is equal to 3, and the value of (*nanwu*-1) or (*qingnian*-2) is 1. Based on the statistical results, the Traffic Bureau should arrange more patrolmen on road *shisan* to foster better traffic and pedestrian safety.

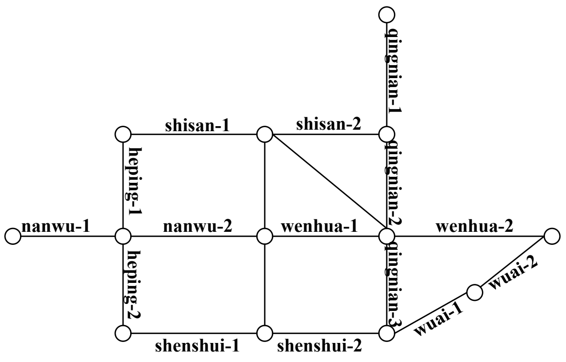


Fig. 2. road network of Shenyang in China

**Advertisement putting on public transportation tools** The public transportation tools such as buses and metros are the most cost-effective media for advertising, to achieve the required coverage and number of exposures in a target audience. The most obvious method is through billboards. For example, indoor billboards intend to attract the interest of people traveling on roads or public transportation. Of course these billboards are generally located in high traffic areas, which can be directly reflected by our sequence range query. That is, the public transportation system records the boarding and alighting stop s/stations of buses/metros of passengers by reading the smart transportation cards, based on which trajectories can be recovered. By summing up these trajectories, pathes with high traffic can be found. Obviously, putting advertisements onto buses/metros in these pathes can cover a lot of passengers, and then increase the "opportunity to see" (OTS) [**?**, **?**].

**Train manufacturing** One component of a train needs to be processed by several departments in the train manufacturing [**?**]. The number of components that one department processes indicates the workload. Usually, a big department consists of several sub-departments. The summation for the numbers from sub-departments, i.e., sequence range query, reflects the workload of this big department. The boss can make the wages for employees based on the summations. And in the real application, considering the heavy workload of some sub-departments, the subprocesses of some components are outsourced to the third parties. If we use the traditional sequence query semantic, it returns the number of components which are processed by sub-departments jointly. The result cannot reflect the true workload of the big department.

**Internet traffic analysis** administrator of a website can get the visiting information (urls) of visitors from the logs or cookies. And the structure of website decides the partial order among the urls. The result of sequence range query along one path indicates the hits of the urls on this path, which reflects the popularity of urls. Furthermore, this information can be used to adjust the structure of the website, in order to improve the user experience.

## 3.2 Problem Formulation

Given a sequence database $D = \{s_1, s_2, \dots, s_n\}$, another database $D' = \{s_1, ..., s_{t-1}, s'_t, s_{t+1}, ..., s_n\}$ is a neighbor database to $D$, if it differs from $D$ on one and only one sequence $s'_t$. Under the DP model, we aim to design a mechanism $\mathcal{M}$ to publish the statistics of the database, with indistinguishable outputs on $D$ and $D'$.

**Definition 2.** ($\varepsilon$-DP [**?**]). *A privacy mechanism $\mathcal{M}$ enforces $\varepsilon$-DP, if for any two neighbor databases $D$ and $D'$, the outputs of $\mathcal{M}$ satisfy* $\Pr[\mathcal{M}(D) \subseteq \text{O}] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{M}(D') \subseteq \text{O}]$.

The positive parameter $\varepsilon$, a.k.a. *privacy budget*, controls the strength of privacy protection. When the output domain of the mechanism $\mathcal{M}$ consists of real numbers, there is a simple solution to achieve $\varepsilon$-DP, based on the concept of *sensitivity*, as defined below.

**Definition 3.** (Sensitivity [**?**]). *Let query $Q$ be modeled as a mapping from databases to a d-dimensional vector of real numbers, i.e. $Q(D) \in \mathbf{R}^d$. The sensitivity is the maximal change over the outputs on any pair of neighbor databases under $L_1$ norm distance, i.e., $S(Q) = \max_{D,D'} |Q(D) - Q(D')|_1$.*

In our sequence database, each sequence $s_i$ contributes unit weight to at most $|s_i| - 1$ counts in the queries. It means that the sensitivity of any sequence range query is $\Delta$ -1, where $\Delta$ is the upper bound for the lengths of sequences in the database and is regarded as a prior knowledge without privacy.

**Theorem 1.** (Laplace Mechanism[**?**]) *A randomized algorithm $\mathcal{M}$ enforces $\varepsilon$-DP, by adding independent Laplace noise with zero mean and $\frac{S(Q)}{\varepsilon}$ scale to query output $Q(D)$.*

In the rest of this section, we use $Lap\,(\lambda)$ to denote the random variable following zero-mean Laplace distribution of scale $\lambda$. The output of Laplace mechanism can thus be formulated as $\hat{Q}(D) = Q(D) + Lap\left(\frac{S(Q)}{\varepsilon}\right)$.

Assume a mechanism $\mathcal{M}(D)$ operates by independently running two algorithms $\mathcal{M}_1(D)$ and $\mathcal{M}_2(D)$. If $\mathcal{M}_1$ satisfies $\varepsilon_1$-DP and $\mathcal{M}_2$ satisfies $\varepsilon_2$-DP, $\mathcal{M}$ must enforce $(\varepsilon_1 + \varepsilon_2)$-DP. This is known as the *composition property* of DP, which is commonly used to design complex differentially private data processing algorithms.

The goal of our study is to minimize the error on sequence range queries, while enforcing DP. In particular, we use mean squared error as the major metric of sequence range query quality.

**Definition 4.** (Mean Squared Error). *Given a group of queries $\mathbf{Q} = \{Q_1, \dots, Q_l\}$, with exact results $\{Q_1(D), \dots, Q_l(D)\}$ and noisy results $\{\hat{Q}_1(D), \dots, \hat{Q}_l(D)\}$ outputted by differentially private mechanism $\mathcal{M}$, the Mean Squared Error (MSE) is formalized as $\frac{\sum_{j=1}^{l}(Q_j(D) - \hat{Q}_j(D))^2}{l}$.*

In conclusion, we aim to design a new publication mechanism to process range queries, over sequence data following an event graph. Our mechanism tends to minimize the mean squared error on the query results, while preserving DP with a specified privacy budget parameter $\varepsilon$.

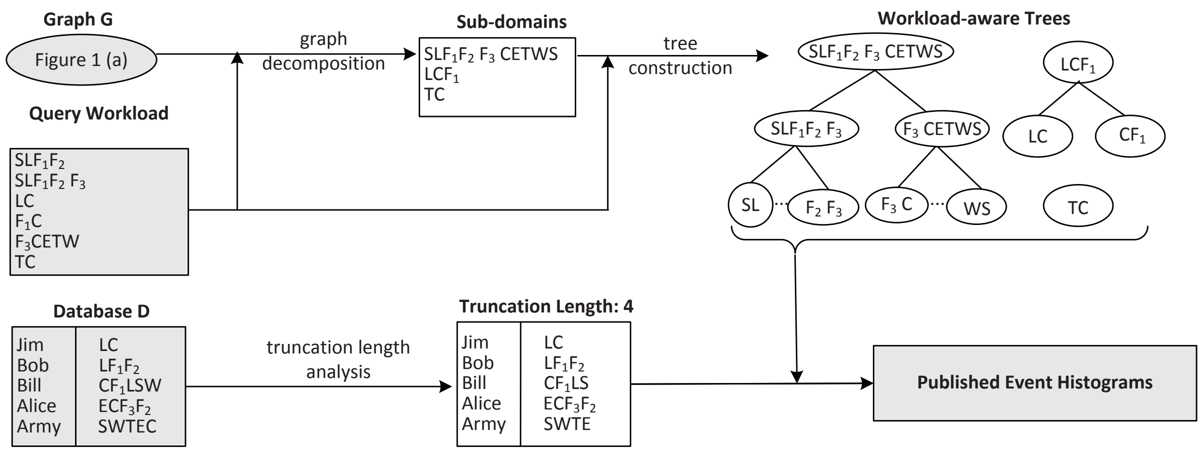


Fig. 3. The Framework of Event Histograms Publication

### 3.3 Algorithm Framework

In this paper, we propose a three-step framework to tackle the problem of histogram publication on sequence data. In Figure 3, we present an example to illustrate the framework with the graph structure and database in Figure 1. In the following, we list the major functionality and output of the steps. The technical details of these steps will be discussed in the following.

**Truncation Optimization:** To reduce the sensitivity of the sequence database, our method collects certain basic statistics of the database $D$. Based on noisy statistics with appropriate Laplace noise, the system chooses a truncation length which is expected to deliver better query accuracy. In Figure 3, for example, the algorithm decides to truncate all sequences to length 4. After the truncation, a new truncated database is generated and fed into the following algorithms in the framework.

**Event Graph Decomposition:** Graph decomposition aims to partition the original event graph $G$ into a group of sub-domains, or paths in other words. This decomposition is run based on the information in the event graph as well as the sample query workload. The decomposition is optimized by minimizing the number of sub-queries needed to evaluate all the sample queries. In Figure 3, the original graph in Figure 1 is partitioned into three sub-domains.

**Index Optimization:** Given the groups of sub-domains output from graph decomposition, our framework constructs a hierarchical tree structure based on the distribution of sub-queries on the sub-domains. The result tree structure is combined with the truncated database to generate a group of counts on each node in the tree structure, corresponding to the count aggregation of a sequence in the sub-domains. After injecting noises on all counts over the tree, a consistency enforcement algorithm is applied to ensure the counts are consistent with each other. The counts on the bottom levels of the trees are published as the result.

## 4 Truncation Optimization

The first technique used in the event histogram publication is *record truncation*, which can cut down the sensitivity of original sequence database. Although it is already used in [**?**], we reformulate the technique for our problem setting, with rigorous error analysis and automatic truncation length optimization. Before discussing our *record truncation* technique, we introduce how to estimate the upper bound of sequence length $\Delta$ using a differentially private method [**?**] instead of being specified by data holder. Specifically, we allocate a small part of privacy budget $\varepsilon_e = 0.05\varepsilon$ to publish an early version of noisy length statistic vector $\tilde{C} = \{C_1 + Lap(\frac{1}{\varepsilon_e}), ..., C_|E| + Lap(\frac{1}{\varepsilon})\}$, where

$C_i$ denotes the number of sequences containing $i$ edges and $Lap(\frac{1}{\varepsilon_e})$ is the Laplace noise with mean 0 and scale $\frac{1}{\varepsilon_e}$. Accordingly, $\Delta$ is set to the value such that the percentage of the sequences with length no greater than $\Delta$ is at least 85%. After that, all sequences are truncated, by randomly choosing a subsequence of length at most $\Delta$. In the following, $\Delta$, the new database $D$ and remaining privacy budget $\varepsilon = \varepsilon - \varepsilon_e$ are fed up into the techniques including *record truncating*, *event graph decomposition* and *index optimization* to support the data publication for partially ordered domains

In Algorithm 1, we list the process of the basic algorithm. The algorithm firstly partitions the original privacy budget into two parts, i.e., $\varepsilon_1$ and $\varepsilon_2$. Given the partition result, $\varepsilon_1$ is used to protect the sensitive information in length statistics, the vector with the number of sequences of particular lengths, and $\varepsilon_2$ is spent on publishing noisy counts on the number of sequences on all edges. The algorithm estimates the preferable truncation length $L$ based on the noisy length statistics $\widehat{C}$. All sequences are truncated, by randomly choosing a subsequence of length at most $L$. Finally, the algorithm releases the counts on the truncated subsequences over all edges, denoted by $\hat{x}_e$ for every $e \in E$, as shown in the algorithm.

**Algorithm 1:** *Truncation-Based* Algorithm

```
Input: event graph $G(V, E)$, database $D$, privacy budget $\varepsilon$, upper bound of
       sequence lengths $\Delta$
1 Partition $\varepsilon$ into two parts $\varepsilon_1$ and $\varepsilon_2$
2 Collect the statistics vector of noisy sequence length $\widehat{C} = \left(\widehat{C}_1, \dots, \widehat{C}_\Delta\right)$
    under $\varepsilon_1$-DP.
3 Choose $L$ as the truncation length which makes the value of Formula 1
    minimum, where we use $\widehat{C}$ instead of $C$.
4 Publish the noisy counts of all edges with $L$, $\{\hat{x}_e\}$, under $\varepsilon_2$-DP.
```

## 4.1 Truncation Length Analysis

The key of the algorithm is to find optimal truncation length $L$ with an error estimation model, which is not supported by any existing truncation approach, e.g. [?]. The error estimation model is based on the length statistics vector $C = (C_1, \dots, C_\Delta)$, with $C_j$ as the number of sequences in $D$ of length $j$. Apparently, the sensitivity of the vector is exactly 1. Our algorithm thus generates noisy counts with privacy budget $\varepsilon_1$, by inserting Laplace noise of scale $1/\varepsilon_1$ into each count, i.e.

$$\widehat{C} = (C_1 + Lap\left(1/\varepsilon_1\right), \dots, C_\Delta + Lap\left(1/\varepsilon_1\right))$$

To find the length $L$ for truncation, it is important to estimate the error incurred by the truncation operation as well as the following noise injection operation. While an exact error estimation in terms of truncation length $L$ is generally difficult, the following analysis derives an upper bound on the error for optimization. Particularly, the expected error on all the counts can thus be simplified as $\mathbf{E}\left[\sum_{i=1}^{|E|}(\hat{x}_i - x_i)^2\right]$. Accordingly, Lemma 1 provides an efficient way of a tight upper bound evaluation.

**Lemma 1.** *Given* $\{C_1, \dots, C_\Delta\}$, *the expectation of* $\sum_{i=1}^{|E|}(\hat{x}_i - x_i)^2$ *is upper bounded as*

$$\mathbf{E}\left[\sum_{i=1}^{|E|}(\hat{x}_i - x_i)^2\right] \leq \frac{2(L-1)^2|E|}{\varepsilon_2^2} + \left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2 \tag{1}$$

*Proof.* Based on the workflow of Algorithm 1, the noise on each $\hat{x}_i$ consists of two parts, i.e. $\hat{x}_i - x_i = n_i - f_i$, where $n_i$ is the Laplace noise and $f_i$ is the frequency loss due to the truncation operation. We thus decompose the expected error into two parts, as

$$\mathbf{E}\left[\sum_{i=1}^{|E|}(\hat{x}_i - x_i)^2\right] = \mathbf{E}\left[\sum_{i=1}^{|E|}(n_i - f_i)^2\right] = \frac{2(L-1)^2|E|}{\varepsilon_2^2} + \sum_{i=1}^{|E|} f_i^2$$

The first component in the equation above depends on $L$, $|E|$ and $\varepsilon_2$ only. In the following, we build a connection between the second component and sequence length statistics $\{C_i\}$. It is obvious that $\sum_{i=1}^{|E|} f_i^2 \leq \left[\sum_{j=1}^{\Delta-L} jC_{L+j}\right]^2$. This finishes the proofs. $\square$

Based on the noisy vector $\widehat{C}$ of $C$, Lemma 1 is used to estimate the maximal error incurred by truncation under DP. It implies the tradeoff between the Laplace noise scale and the frequency loss. With the increase of $L$, the scale of Laplace noise grows but the frequency losses are reduced. In our *truncation-based* algorithm, the system simply iterates on all possible truncation length $L$s, evaluates the error upper bound by Lemma 1 and returns the $L$ with the minimal error estimation. Then for the sequence whose length is bigger than $L$ in the database, we choose one subsequence with the length of $L$ from the sequence randomly. An important advantage of the estimation is the utilization of sequence length statistics, which is perturbed with a small amount of noise due to the constant sensitivity under the DP framework.

## 4.2 Privacy Budget Assignment

This part discusses how to partition the privacy budget $\varepsilon$ into $\varepsilon_1$ and $\varepsilon_2$, in order to optimize the performance of the *truncation-based* algorithm. By using $\{\widehat{C}_i\}$ instead of $\{C_i\}$ in the formula derived by Lemma 1, additional noise is injected into the upper bound estimation on $\mathbf{E}\left[\sum_{i=1}^{|E|}(\hat{x}_i - x_i)^2\right]$, implied by Theorem 2.

**Theorem 2.** *By replacing $\{\widehat{C}_i\}$ with $\{C_i\}$, the upper bound estimation is updated as*

$$\mathbf{E}\left[\sum_{i=1}^{|E|}(\hat{x}_i - x_i)^2\right] \leq \left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2 + \frac{A_1}{\varepsilon_1^2} + \frac{A_2}{\varepsilon_2^2}$$

*where* $A_1 = \frac{(\Delta-L)(\Delta-L+1)(2\Delta-2L+1)}{3}$, $A_2 = 2(L-1)^2|E|$.

*Proof.* Let $\delta_i$ denote the Laplace noise of $\widehat{C}_i$, where $\delta_i = \widehat{C}_i - C_i$. Obviously, when $i \neq j$, $\mathbf{E}\delta_i = 0$ and $\mathbf{E}\delta_i\delta_j = 0$. We have the following formula:

$$\mathbf{E}\left(\sum_{j=1}^{\Delta-L} j\widehat{C}_{L+j}\right)^2 = \mathbf{E}\left(\sum_{j=1}^{\Delta-L} jC_{L+j} + \sum_{j=1}^{\Delta-L} j\delta_{L+j}\right)^2$$
$$= \mathbf{E}\left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2 + \mathbf{E}\left(\sum_{j=1}^{\Delta-L} j\delta_{L+j}\right)^2$$
$$= \mathbf{E}\left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2 + \frac{(\Delta-L)(\Delta-L+1)(2\Delta-2L+1)}{3\varepsilon_1^2}$$

We reach the conclusion of this theorem by combining the equation above with Lemma 1. □

In the error analysis of Theorem 2, the upper bound consists of three parts. The first part depends on $L$, $\Delta$ and database, and the other two parts depend on $L$, $\Delta$, $\varepsilon_1$ and $\varepsilon_2$. But when we partition $\varepsilon$, the truncation length $L$ is unknown. Hence, we compute $\varepsilon_1$, $\varepsilon_2$ by minimizing the expected upper bound for different $L$s, which is described as:

$$\text{Minimize: } \mathbf{E}\left[\left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2 + \frac{A_1}{\varepsilon_1^2} + \frac{A_2}{\varepsilon_2^2}\right]$$
$$\text{s.t. } \varepsilon_1 + \varepsilon_2 = \varepsilon$$

**Theorem 3.** *To minimize the upper bound in Theorem 2 for different Ls, $\varepsilon_1$ and $\varepsilon_2$ are*

$$\varepsilon_1 = \frac{\varepsilon\sqrt[3]{\sum_{L=2}^{\Delta} A_1}}{\sqrt[3]{\sum_{L=2}^{\Delta} A_1} + \sqrt[3]{\sum_{L=2}^{\Delta} A_2}}, \varepsilon_2 = \varepsilon - \varepsilon_1.$$

*Proof.* Obviously, we have the following formula:

$$\mathbf{E}\left[\left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2 + \frac{A_1}{\varepsilon_1^2} + \frac{A_2}{\varepsilon_2^2}\right]$$
$$= \frac{\sum_{L=2}^{\Delta}\left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2 + \sum_{L=2}^{\Delta}\left(\frac{A_1}{\varepsilon_1^2} + \frac{A_2}{\varepsilon_2^2}\right)}{\Delta - 1}$$

The values of $\sum_{L=2}^{\Delta}\left(\sum_{j=1}^{\Delta-L} jC_{L+j}\right)^2$ and $\Delta$ are independent of $L$, $\varepsilon_1$ and $\varepsilon_2$. So the question is transformed to: minimizing $\left(\sum_{L=2}^{\Delta}\left(\frac{A_1}{\varepsilon_1^2} + \frac{A_2}{\varepsilon_2^2}\right)\right)$ under the constraint $\varepsilon_1 + \varepsilon_2 = \varepsilon$. Simply, by the Lagrange multipliers method, we get $\varepsilon_1$ and $\varepsilon_2$. □

## 4.3 Discussion the boundary of our technique

In the above discussion, *truncation optimization* focuses on dealing with the sequences which are pathes along the given graph. In fact, for the sequences consisting of edges in the graph but not on one path, *truncation optimization* is also suitable with slight modification. It is because *truncation optimization* is designed based on the counts of edges, independent of the correlation among adjacent edges. We slightly modify the technique like this: after truncating length $L$ is derived, all sequences are truncated, by randomly choosing at most $L$ edges from the original sequence to form a new sequence instead of choosing a sub-sequences of length at most $L$. In the following, *event graph decomposition* and *index optimization* are proposed based on the query workload and graph, unrelated with the sequence database. So the techniques put forward in this paper can tackle the case where the sequences consisting of edges in the graph but not on one path.

## 5 Event Graph Decomposition

Although truncation is useful to directly cut off the sensitivity of the sequences and reduce the scale of noises on individual counts under the DP framework, the number of sub-ranges used to answer a range query remains large. For traditional range queries on a normalized domain, hierarchical reorganization on the counts, e.g. [**?**], is proven effective on reducing the number of involved counts and minimizing the final noises for range queries. However, it is difficult to combine existing optimization techniques, mainly due to the complex graph structures on the sequences. This section devotes to designing a decomposition algorithm over the graph structure, in order to transform the event graph into multiple paths, i.e., sub-domains. After the transformation, conventional DP methods can be immediately applied over each sub-domain, by projecting the truncated sequences and queries onto the sub-domains.

Specifically, given a query set $\mathbf{Q} = \{Q_1, \ldots, Q_l\}$ and a graph structure $G(V, E)$, the goal of the decomposition is to construct a group of sub-domains, i.e. a non-empty set of sub-graphs $\mathbf{C} = \{G_1(V_1, E_1), \ldots, G_m(V_m, E_m)\}$. It is a valid decomposition, if $V_j \subseteq V$ for any $1 \leq j \leq m$, $E_j \cap E_l = \emptyset$ when $j \neq l$, and $\cup_j E_j = E$. Moreover, each $G_j(V_j, E_j)$ is supposed to define a *path sub-domain* over the edges in $E_j$, as defined below.

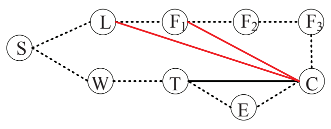
**Definition 5.** *(*Path Sub-domain*) A subgraph $G_j(V_j, E_j)$ defines a path sub-domain, if all the edges in $E_j$ are able to form one path without repetitive edges.*

To protect more order information in the set of sub-domains, the event graph is decomposed only at the *event pivot*s, which are defined in Definition 6. In the following, we use $D_v$ to denote edges connected to event vertex $v \in V$.

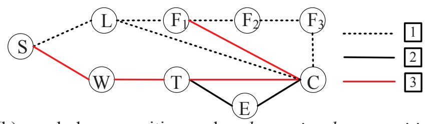
**Definition 6.** Event Pivot *denotes a vertex $p$ of which the degree is more than 2, i.e., $|D_p| > 2$.*

It is obvious that different decomposition results lead to various performance of optimization methods over the domains. In Figure 4, for example, we present two different decompositions over the sightseeing path graph described in Figure 1, using lines with different styles to represent the result sub-domains. Given a query set with 6 queries, as listed in the first column of the table, the queries are partitioned into different sub-queries, depending on the sub-domains. If the decomposition is carefully designed, most of the queries could be processed within a single sub-domain. The decomposition on the left, denoted as *min-cut*, generates 7 sub-queries for all original queries, while the decomposition on the right, denoted as *alternative*, needs to process 11 sub-queries.

This example motivates us to design an optimization framework for event graph decomposition, to minimize the number of sub-queries to process for all queries in the query workload. The principle ensures one query to be answered only on a few sub-domains,



(a) graph decomposition under *min-cut*



(b) graph decomposition under *alternative decomposition*

| *Query Set* | weight | *Query Decomposition* | | | |
|---|---|---|---|---|---|
| | | Min-cut | | Alternative | |
| *WTE* | *$SLF_1$* (1) | *WTE* | $\boxed{1}$ | *WT+TE* | $\boxed{3}$ $\boxed{2}$ |
| *$F_3CE$* | *$LF_1F_2$* (1) | *$F_3CE$* | $\boxed{1}$ | *$F_3C+CE$* | $\boxed{1}$ $\boxed{2}$ |
| *$LCF_1$* | *WTE* (2) | *$LCF_1$* | $\boxed{3}$ | *$LC+CF_1$* | $\boxed{1}$ $\boxed{3}$ |
| *$SLF_1F_2$* | *WTC* (1) | *$SLF_1F_2$* | $\boxed{1}$ | *$SLF_1F_2$* | $\boxed{1}$ |
| *$WTECF_3$* | *$F_3CE$* (2) | *$WTECF_3$* | $\boxed{1}$ | *$WT+TEC+CF_3$* | $\boxed{3}$ $\boxed{2}$ $\boxed{1}$ |
| *WTC* | *$LCF_1$* (1) | *WT+TC* | $\boxed{1}$ $\boxed{2}$ | *WTC* | $\boxed{3}$ |

(c) how to answer the query workload under different decomposition ways

Fig. 4. Decomposition of the event graph

which provides more optimization room for the path sub-domains in Section 6. In the following, we show this principle is equivalent to maximizing the number of queries which are not split at *event pivot*s. Firstly, Lemma 2 shows that it is always beneficial to combine two sub-domains, if their paths are combinable.

**Lemma 2.** *Given two sub-domains $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ from a valid decomposition $\mathbf{C}$, if $E_1 \cup E_2$ forms a path, it always reduces the number of sub-queries by combining $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$.*

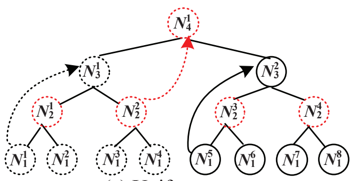
Given an *event pivot* $p$, $(v_r, p) \in D_p$ and $(v_t, p) \in D_p$, Lemma 2 implies that it is always beneficial by combining certain $v_r$ with $v_t$, such that query sequences covering $(v_r, p)$ and $(p, v_t)$ are not split into sub-queries. Here, we use $\omega(v_r, p, v_t)$ to denote the query workload over $(v_r, p, v_t)$, i.e. the exact number of queries in $\mathbf{Q}$ with both edges $(v_r, p)$ and $(p, v_t)$. For a specific decomposition $\mathbf{C} = \{G_1, G_2, \dots, G_m\}$ and a query set $\mathbf{Q} = \{Q_1, \dots, Q_l\}$, the total number of sub-queries for all these $l$ queries is:

$$|\mathbf{Q}| + \sum_p \left( \sum_{(v_r,p,o_t)} \omega(v_r, p, o_t) - \sum_{G_j \in \mathbb{C}} \sum_{(v_r,p,o_t) \in G_j} \omega(v_r, p, o_t) \right)$$

Since both $|\mathbf{Q}|$ and $\sum_p \sum_{(v_r,p,o_t)} \omega(v_r, p, o_t)$ are constants, the minimization on the number of sub-queries with respect to $\mathbf{Q}$ is achieved by the following optimization program on decomposition $\mathbf{C}$:

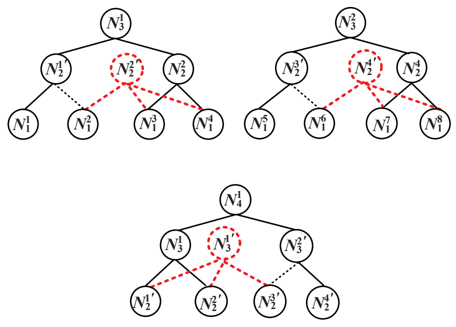
$$\text{Maximize:} \sum_p \sum_{G_j(V_j,E_j) \in \mathbf{C}} \sum_{(v_r,p,o_t) \in G_j} \omega(v_r, p, o_t) \tag{2}$$

From the optimization program above, we can see that the optimal decomposition is achieved by independently combining the edges in the sub-domains for each *event pivot* $p$. Combined with the implication of Lemma 2, the optimization framework runs maximum matching over the edges of each *event pivot* in order, as shown in Algorithm 2. Then based on the matching results, *min-cut* searches the connected paths as sub-domains. Note that by Algorithm 2, if one sub-domain $G'(V', E')$ consists of one cycle, we split the cycle at node $p$, where $\omega(v_r, p, o_t)$ is minimum. Here, $p, v_r, o_t \in V'$. Besides, Algorithm 2 can also be applied for the directed graph decomposition with tiny modification. Specially, instead of Line 4-5, for every event pivot, we need to construct its in-degree vertexes set $I$ and out-degree vertexes set $O$. And then the maximum matching is run based on the elements in $I$ and $O$ to obtain the event edge combination table $\mathbf{T}$.


(a) Uniform tree

| *Query* | *Combination* | |
|---|---|---|
| | (1),(2,3,4) | (1,2),(3,4) |
| 2,3,4<br>2,3,4,5<br>1,2,3,4 | (2,3,4)<br>(2,3,4),(5)<br>(1),(2,3,4) | (2),(3,4)<br>(2),(3,4),(5)<br>(1,2),(3,4) |

(b) The way to answer the query workload under different combinations


(c) Bottom-up phase

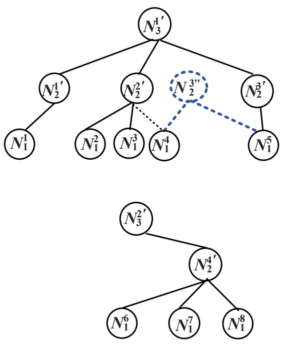

(d) Top-down phase

Fig. 5. The construction of workload-aware structure

**Algorithm 2:** *min-cut* Algorithm

```
Input  : event graph $G(V,E)$, query set $\mathbf{Q}$
Output: sub-domain set $\mathbf{C}$
1  Initialize an empty decomposition $\mathbf{C} = \emptyset$;
2  for each node $p$ in $V$ do
3      if $|D_p| > 2$ then
4          Construct $I = \{(v_r, p) \mid v_r \in D_p\}$;
5          Run maximum matching among elements in $I$ with $\omega(v_r, p, v_t)$
             as the weights, where $(v_r, p)$ and $(v_t, p) \in I$;
6      Store each pair $\{(v_r, p), (v_t, p)\}$ in maximum matching in table $\mathbf{T}$;
7  while $E \neq \varnothing$ do
8      Choose one edge $(v_r, v_t)$ and remove it from $E$;
9      Initialize one sub-domain $G'(V', E')$ with $(v_r, v_t)$;
10     Search $G$ in depth-first from $v_t$ using $\mathbf{T}$ at event pivots and get new
         vertices and edges $G_\triangle = (V_\triangle, E_\triangle)$;
11     Update $G'$ as $V' = V' \cup V_\triangle$ and $E' = E' \cup E_\triangle$;
12     Update G as $E = E - E'$ ;
13     Execute Line 10-12 with vertex $v_r$;
14     Add $G'$ to $\mathbf{C}$;
15 Return $\mathbf{C}$ as the decomposition result;
```

## 6 Index Optimization

After decomposing event graph $G(V,E)$ into path sub-domains, existing optimization methods for DP are directly applicable. Nevertheless, most of them do not consider the non-uniform query workload over the domain. Given a query workload $\mathbf{Q} = \{Q_1, Q_2, \ldots, Q_l\}$ and sub-domains $\mathbf{C} = \{G_1, G_2, \ldots, G_m\}$, each query $Q_j$ is correspondingly projected onto $m$ sub-queries, e.g. $\{Q_j^1, \ldots, Q_j^m\}$. The result of $Q_j$ is the aggregation over all these sub-queries, i.e., $Q_j(D) = \sum_{t=1}^{m} Q_j^t$. Therefore, the query workload on a particular sub-domain $G_t$ consists of sub-queries $\{Q_1^t, Q_2^t, \ldots, Q_l^t\}$, probably with some empty sub-query $Q_j^t$.

We focus on hierarchical optimization techniques [?, ?], which generally group the edges in the sub-domain at different granularities with a tree structure. In Figure 5(a), for example, we present a typical binary tree index over a normalized sub-domain with 9 concrete events (8 edges). Each node in the tree corresponds to an interval over the sub-domain, associated with a noisy number as the summation over the counts within the interval. Given a sub-query over a specified interval on the sub-domain, the tree structure processes it in a top-down manner, which tries to use noisy numbers on high-level nodes as much as possible and pushes the query downward if it cannot be fully answered. Compared with other optimization methods, e.g. [?, ?], the construction of tree-based indices is usually cheap in computation cost and the accuracy performance is mostly competitive. Given a sub-query on the sub-domain, the accuracy of the query result is improved, if a few noisy counts on the tree structure are used to answer the query.

A straightforward baseline solution is to build a tree structure with fanout $k_i$ for each subdomain $G_i(V_i, E_i)$, with $k_i$ calculated based on the existing study in [?]. This results in a global sensitivity in terms of the forest structure as $H_{max} \times (L-1)$, in which $H_{max}$ is the maximal height of all trees and $L$ is the truncation length. Note that the noise scale on-

Assumes that all the queries are submitted with the same probability. Ref. [?] gives the expression for error of all the queries which only depends on $k$. The value of $k$ is derived to minimize the error of all the queries.

ly depends on the maximal tree height $H_{max}$. It thus makes sense for a sub-domain $G_i(V_i, E_i)$ to increase tree height to $H_{max}$ by re-adjusting the fanout factor $k_i$ to $k_i'$. In the rest of the section, we use $K^{'} = \{k_i'\}$ to record all adjusted fanout factors of the hierarchical structures of the sub-domains.

To simplify the notation in the rest of the section, for a tree structure, we use $N_i^j$ to denote the $j$-th node on the $i$-th level of the tree, as well as the count kept at the node. On the bottom level (first level), each node $N_j^1$ bears the count for one edge in the sub-domain, while the count on a high level node is the sum of the counts on its children nodes. Let $\mathbf{N}(i)$ denote the number of nodes in the $i$-th level and $CN_i^j$ indicate the children node set of $N_i^j$. $\omega_i^j$ is the number of queries covering the interval of the node $N_i^j$.

The structure of the tree may have huge impact on the querying performance, especially with non-uniform workload. For instance, Figure 5(a) shows a conventional binary tree used by [**?**, **?**]. In this standard structure, $(N_1^1, N_1^2)$ are children of $N_2^1$, and $(N_1^3, N_1^4)$ are children of $N_2^2$. Given the query set in Figure 5, these three queries are answered by using different counts 7 times in total. If we re-construct the tree, by re-assigning $N_1^2$ to $N_2^2$, the frequency of count usage declines to 5, definitely improving the accuracy of results due to less noise. This example motivates us to optimize the tree structure in terms of the query workload. In the following, we present a local search strategy, which iteratively updates the tree structure to improve the performance on the sample queries.

## 6.1 Workload-Aware Structure Search

Based on the initial uniform trees, with fanout factors in $K'$, our *workload-aware structure search* tries to improve the hierarchial structure by re-combining the nodes in the trees based on the sample query workload. The result trees can thus be non-uniform and the fanouts of the nodes can be completely different. The structure search is done with two phases, including *bottom-up* phase and *top-down* phase. Specially, the *bottom-up* phase adjusts the parent node assignment (i.e., combination) from bottom levels to top levels. After finishing *bottom-up*, our algorithm calls another round of *top-down* updates on the tree structure, to further improve the performance. The procedure is shown in Algorithm 3. In the following, we introduce the *bottom-up* phase and *top-down* phase in detail.

**Bottom-Up Phase:**

The key of the algorithm in this phase is the re-assignment mechanism for all nodes on the $i$-th level, with adjustments on $\{CN_{i+1}^j\}$. On each level $i$, the algorithm first groups all nodes on the level based on their grandparent nodes on the tree. Each node group consists of at most $k^2$ nodes, with $k$ as the fanout of the original uniform tree structure. In Figure 5(a), for example, there exist two nodes on the third level, which results in two groups of nodes, i.e. $\{N_1^1, N_1^2, N_1^3, N_1^4\}$ and $\{N_1^5, N_1^6, N_1^7, N_1^8\}$ for $i = 1$. Since there is only one node on the fourth level, there is only one node group $\{N_2^1, N_2^2, N_2^3, N_2^4\}$ for $i = 2$. On each group, our updating algorithm tries to re-assign nodes to a different parent node, in order to minimize the frequency of using the nodes for query processing. After partitioning $k^2$ nodes into $k$ parts, each part is assigned to one node on level $i + 1$, i.e. $CN_{i+1}^j$. In Figure 5(c), for the group $\{N_1^1, N_1^2, N_1^3, N_1^4\}$, the re-combination with $\{N_1^1\}$ and $\{N_1^2, N_1^3, N_1^4\}$ can minimize such frequency of usage. We thereby move $N_1^2$ from $CN_2^1$ to $CN_2^2$. Similarly, $N_1^6$ is re-assigned to $CN_2^4$.

To formulate the parent re-assignment problem by an optimization program on $\{CN_{i+1}^j\}$, we model the effects of re-assignments as follows. We define the utility of a node by its frequency of usage in query processing, which is equal to the difference between the number of queries covering the interval of the node and that of its parent node. If the number of queries covering node $N_i^y$ is $\omega_i^y$ and $N_{i+1}^l$ is its parent, the utility of $N_i^y$ is thus computed as $\omega_i^y - \omega_{i+1}^l$. Given a node group on the $i$-th level with grandparent node $N_{i+2}^j$, the total utility of all nodes on the $i$-th level and the $(i+1)$-th level within the group is

$$\sum_{N^x_{i+1} \in CN^j_{i+2}} \Big( \sum_{N^y_i \in CN^x_{i+1}} (\omega^y_i - \omega^x_{i+1}) \Big) + \sum_{N^x_{i+1} \in CN^j_{i+2}} (\omega^x_{i+1} - \omega^j_{i+2})$$

**Algorithm 3:** *Workload-Aware* Algorithm

```
Input : sub-domain set $\mathbf{C}$, fanout set $K'$
Output: $optiTreeSet$
1  Initialize an empty tree set $optiTreeSet = \emptyset$
2  for each $G_i(V_i, E_i) \in \mathbf{C}$ do
3      Build an uniform $K'_i$-tree $\mathbf{T}$ for $G_i(V_i, E_i)$
4      for each level $l \in \mathbf{T}$ bottom-up do
5          for each node $N^j_l \in \mathbf{N}(l)$ do
6              Construct set $g$ by $N^j_l$'s grandson nodes
7              run dynamic programming for $g$ to obtain $|CN^j_l|$ new
                 children nodes of $N^j_l$
8      for each level $l \in \mathbf{T}$ top-down do
9          for each node $N^j_l \in \mathbf{N}(l)$ do
10             if interval of $N^j_l$ is changed then
11                 Construct set $g$ by $N^j_l$'s grandson nodes
12                 run dynamic programming for $g$ to obtain $|CN^j_l|$ new
                     children nodes of $N^j_l$
13     Add $\mathbf{T}$ to optiTreeSet
14 return optiTreeSet
```

The parent re-assignment problem is thus modeled to maximize the utilities by choosing $CN^x_{i+1}$ carefully for every node $N^x_{i+1}$ on level $i+1$. By simple algebraic operations, it is not difficult to derive the following maximization problem, which simply implies that a node $N^x_{i+1}$ with a higher usage frequency is expected to cover more children nodes on the $i$-th level.

$$Maximize: \sum_{N^x_{i+1} \in CN^j_{i+2}} (|CN^x_{i+1}| - 1) w^x_{i+1} \quad (3)$$

To find such optimal assignment, we run dynamic programming for every node group on the $i$-th level, together with their potential parent nodes on level $i+1$. Since there are at most $k^2$ nodes in each node group, the computation complexity is $O(k^5)$. Obviously, there are at most $\frac{1-k^{\log_k |E_i|-1}}{1-k}$ groups of nodes in the initial tree structure, where $E_i$ is the edge set of the corresponding sub-domain $G_i$. The total computational complexity of the bottom-up phase is thus $O(k^3|E_i|)$.

**Top-Down Phase:**

In the bottom-up phase, children nodes are re-assigned to new parents without considering any other nodes in the tree structure. When their grandparent nodes are updated later, their previous parent assignment may not be optimal any more. This intuition inspires us to conduct another round of top-down updates on the tree structure. For nodes on the $i$-th level, our method first checks if any of their grandparent node has changed. If a grandparent change is detected, our algorithm runs again the dynamic programming for better parent assignment. In Figure 5(d), for example, the interval covering node $N^1_3$ is now $(N^1_1, \ldots, N^5_1)$ instead of the original $(N^1_1, \ldots, N^4_1)$. Our method thus reconstructs the assignment for $\{N^1_1, N^2_1, N^3_1, N^4_1, N^5_1\}$ and the combination including $\{N^1_1\}$, $\{N^2_1, N^3_1\}$ and $\{N^4_1, N^5_1\}$ exhibits better utilities. As a result, we re-assign $N^4_1$ to $N^3_2$. Despite the scale of nodes and assignments may differ from those in the *bottom-up* phase, the complexity of this step remains $O(k^3|E_i|)$.

Given the updated index tree structure for the sub-domain $G_i$, we finalize the index construction by calculating the counts for each node, based on the truncated sequences and the corresponding intervals of the nodes on the sub-domain. The counts are finally published with the noise, i.e. $\hat{N}^j_i = N^j_i + Lap\left(\frac{(L-1)*H_{max}}{\varepsilon_2}\right)$ for each node in the tree, in which $\varepsilon_2$ is the privacy budget and $L$ is the truncation length as described in Section 4. Apparently, the publication procedure remains consistent with the differential privacy framework, because all computations done by Algorithm 3 in this section do not involve any personal records in the database.

## 6.2 Consistency Enforcement

After noise injection over the counts with the nodes in the tree structure, the consistency of the counts may be broken, as the count of a parent node may not be equal to the sum of the counts on its children nodes. Existing study in [**?**] shows that the consistency is not only important in keeping query result interpretable, but also helpful in improving the querying accuracy. Since the method [?] is designed for ensuring the consistency of tree structure with uniform fanout, it is not applicable to our non-uniform tree structures. Although [?] mentions they have modified the original algorithm and proofs in [?] to work for the non-uniform trees, the modified algorithm is not shown in [?]. We list the consistency enforcement method for non-uniform trees as below.

Provided the noisy counts $\{\hat{N}_i^j\}$ on the tree structure, a consistency enforcement mechanism returns a new group of noisy counts $\{\bar{N}_i^j\}$, which is achieved by minimizing $L_2$ distance with $\hat{N}_i^j$. We acquire the adjustment value $\{\bar{N}_i^j\}$ by performing two operations: (i) computing $z[N_i^j]$ and $a[N_i^j]$ bottom-up based on Formulas 4 and 5 ; (ii) achieving $\bar{N}_i^j$ top-down according to Formula 6. Here, $CN_i^j$ indicates the children nodes set of $N_i^j$. Besides, $PN_i^j$ denotes the nodes on the path from $N_i^j$ to the root of the tree.

$$z[N_i^j] = \begin{cases} \sum\limits_{N_{i'}^{j'} \in PN_i^j} \hat{N}_{i'}^{j'},\ if\ N_i^j\ is\ the\ leaf \\ a[N_i^j] \sum\limits_{N_{i-1}^{j'} \in CN_i^j} z[N_{i-1}^{j'}],\ o.w. \end{cases} \tag{4}$$

$$a[N_i^j] = \begin{cases} 0,\ if\ N_i^j\ is\ a\ leaf \\ \dfrac{1}{\sum_{N_{i-1}^{j'} \in CN_i^j} (1 - a[N_{i-1}^{j'}]) + 1},\ o.w. \end{cases} \tag{5}$$

$$\bar{N}_i^j = \begin{cases} z[N_i^j],\ if\ N_i^j\ is\ the\ root \\ z[N_i^j] + \dfrac{1 - a[N_i^j]}{\sum_{N_i^j \in CN_{i+1}^{j'}} (1 - a[N_i^j])} (\bar{N}_{i+1}^{j'} - \sum\limits_{N_i^j \in CN_{i+1}^{j'}} z[N_i^j]),\ o.w. \end{cases} \tag{6}$$

The process of deducing the value $\{\bar{N}_i^j\}$ is similar to the method [?]. We omit details for brevity.

## 7 Experiment

**Experiment Setup:** We test the algorithms on two datasets *Oldenburg* and *SanJoaquin* in the empirical studies. Based on the road maps of Oldenburg city and San Joaquin City [?], we employ the well recognized spatial-temporal data generator [?] to build commuter trajectories based on the road networks. The result event graph of *Oldenburg* contains 6105 event vertices, 7032 edges. There are 45,184 trajectories generated for *Oldenburg*, whose average length is 8 and maximal length is 75. And *SanJoaquin* has 18,496 event vertices, 24072 edges and 18,170 trajectories whose average length is 24 and maximal length is 82. For all algorithms tested, we manually set 100 as maximal length of all trajectories as background knowledge.

In the experiments, we evaluate the performance of our techniques against existing approaches in the literature. The descriptions on the baseline approaches as well as our own proposed solutions are available in Table 1. The experiments are run over *clustered interval* workload, which randomly selects $1,000$ event pivots in the given graph and generate 5 queries with a given length $c$ for each selected event pivot. Specially, given an event pivot $p$, a concrete query is produced by the following steps: 1) removing two target vertices $t_1$ and $t_2$, from $p$'s neighbors; 2) constructing a path $l_1$ along $t_1$'s neighbors (excluding $p$) with the length of $\frac{c}{2}$; 3) constructing a path $l_2$ as 2); 4) generating a query by concatenating $l_1$ and $l_2$. We choose the query length $c$ from $2, 10, 20, 30, 40, 50$ and generate 6 groups of *clustered interval* workloads. We adopt *mean squared error* of query results as the performance metrics in our experiments.

Generally speaking, we aim to study the impact of privacy budget and query length in the experiments. Three different privacy budgets, i.e. $\{1, 0.1, 0.01\}$, are tested in all groups of experiments. Given a specified query length, we try to evaluate the *clustered interval* workload with the particular length and each experi-

**Table 1.** Algorithms compared in the experiments

| **Algorithm Name** | **Algorithm Description** |
|---|---|
| $dwork$ [**?**] | standard Laplace mechanism by adding Laplace noise to the counts independently |
| $tru$ | truncation-based optimization proposed in Section 4 |
| $basic$ | combining $dwork$ [**?**] with the truncation-based optimization |
| $min$ | min-cut algorithm proposed in Section 5 for graph decomposition |
| $random$ | baseline approach on graph decomposition, which uniformly combines a pair of edges on each event pivot |
| *opt* | local search strategy for tree-based index proposed in Section 6.1 |
| $min$ + *opt* + $tru$ | combining techniques as listed labels |
| $GS$[**?**] | grouping the counts and applying smoothing over the average of the count groups |
| $boost$ [**?**] | conventional tree-based indexing method with a node fanout chosen based on the analysis in [**?**] |
| $FreqItem$ [**?**] | randomly picking $l$ events from one sequence, in which $l$ is set to the value such that the percentage of the sequences with length no greater than $l$ is at least 85% |
| $normalization$[**?**] | utilizing the length of one sequence to normalize these counts it contributes to |
| *n-gram* [**?**] | naive truncation method that is used to publish the counts of sequences, in which we use the default value 20 as the truncation length |
| $HM$ [**?**] | decomposing the event graph using $min$ and assigning weights to hierarchical queries greedily for every sub-domain based on the given workload |

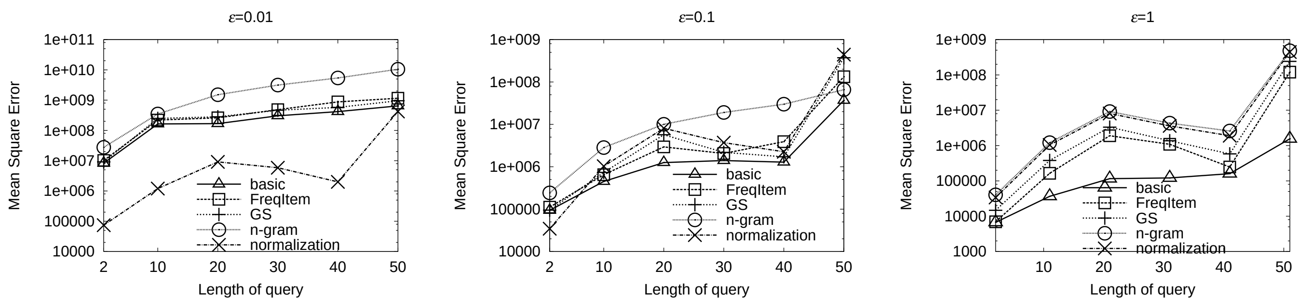


Fig. 6. The effects of sequence truncation on *Oldenburg*

ment is repeated 10 times, with average results reported in this section only. All programs used in the experiments are implemented in C++ and run on a PC with Intel i7 CPU, 4GB RAM and Windows 7 OS.

**Effects of Sequence Truncation:** This part focuses on the truncation technique only. We compare our *basic* method combining the standard Laplace mechanism [**?**] with our truncation technique, against *FreqItem*, *n-gram GS*, and *normalization*. Both *FreqItem* and *n-gram* are truncation-based approaches for reducing the sensitivity, and *GS* is designed to publish the histogram under the case that a user affects several bins. Besides, *normalization* utilizes the length of one sequence to normalize these counts it contributes to. Since these

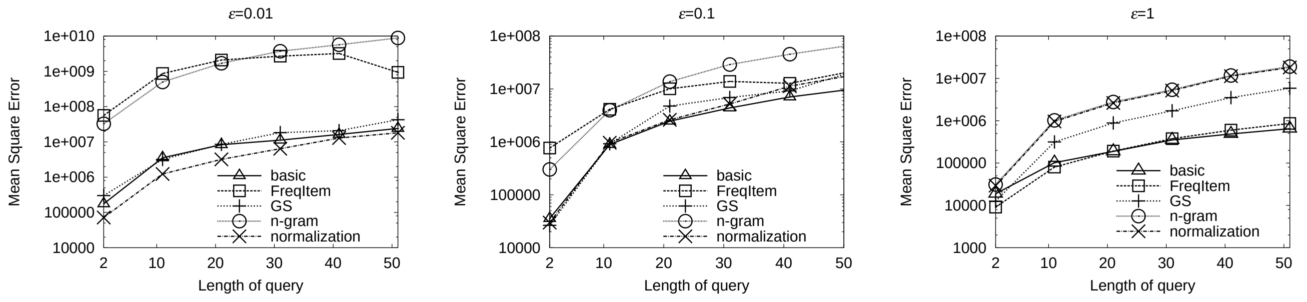


Fig. 7. The effects of sequence truncation on *San Joaquin*

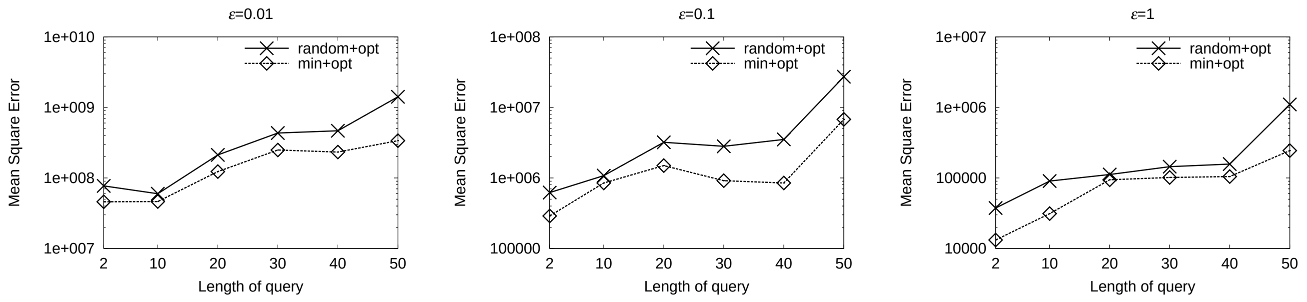


Fig. 8. The effects of event graph decomposition on *Oldenburg*

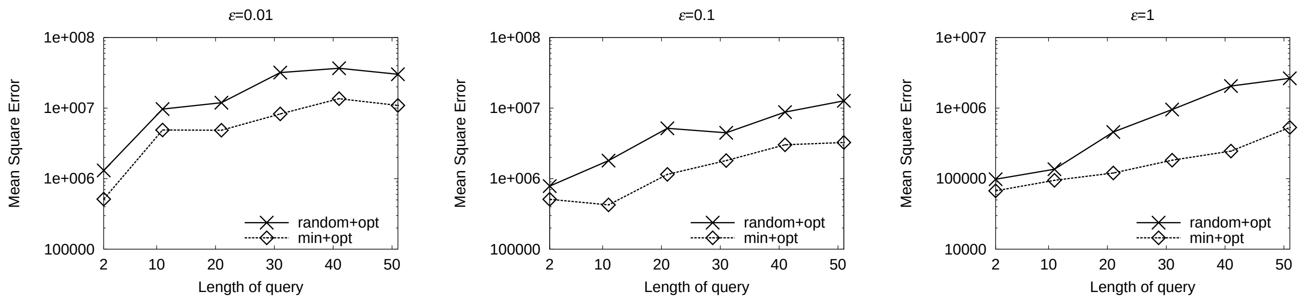


Fig. 9. The effects of event graph decomposition on *San Joaquin*

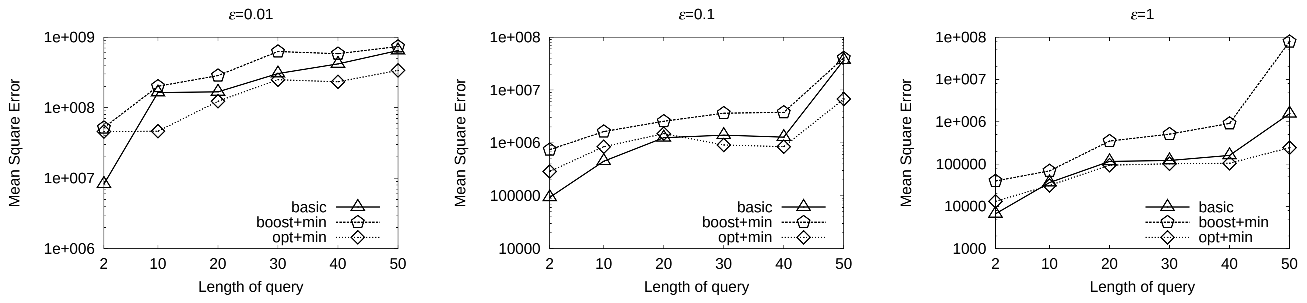


Fig. 10. The effects of tree structure optimization on *Oldenburg*

three methods are well known solutions to high sensitivity, we compare our *basic* with them. As shown in Figure 6 and Figure 7, *basic* outperforms the other methods in most cases. It is because that the truncation length of *n-gram* is manually specified without using actual data distribution. Such manual truncation length cannot guarantee to perform well on an arbitrary dataset. Regarding *FreqItem* approach, although the truncation length of *FreqItem* is calculated based on the dataset, the threshold used in the calculation remains manual and has nothing to do with the privacy budget $\varepsilon$. In contrast, our *basic* automatically chooses a preferable truncation length by a strong theoretical analysis based on the dataset as well as $\varepsilon$. In addition, *basic* outperforms *GS* because *GS* replaces the values of entries in each histogram bin with their average value, to reduce the sensitivity, yet incurring additional errors. Mostly, *basis* shows a superiority to *normalization*. This is because normalizing sequences incurs the bias to the accurate value unavoidably and *normalization* does not consider the tradeoff between the bias and error incurred by noise. However, as shown in the sub-figures ($\varepsilon = 0.01$) of Figure 6 and Figure 7, *normalization* gains a slight superiority to *basic*. The reason is that when $\varepsilon = 0.01$, the bias incurred by normalizing or truncating the sequences is insignificant compared with the error introduced by noise. And $normalization$ has a lower sensitivity than $basic$, which leads to a better performance. Since the effect of truncation is fully validated here, the remaining experiments always include *tru* as a default configuration.

**Effects of Graph Decomposition:** This group of experiments is to investigate the effects of event graph decomposition on querying performance. Basically, we compare our min-cut method *min* against random partitioning algorithm *random*, both of which are accompanied by other optimization techniques for fair comparison. The results are presented in Figure 8 and Figure 9 for two datasets respectively. The figures show that the decomposition method min-cut outperforms the random decomposition method *random* on all datasets and all $\varepsilon$ values. The reason is that *min* protects queries as much as possible from being split at pivot vertices. Hence, *min* enhances the benefits for the subsequent tree structure, as queries with high weights may be covered by a single sub-domain with high probability (discussed in Section 5.1).

**Effects of Tree Structure Optimization:** In this suite of experiments, we test the usefulness of our tree structure optimization method *opt*. Figure 10 and Figure 11 illustrate the mean square errors of three methods, standard Laplace mechanism with truncation technique *basic*, traditional tree-based optimization on sub-domains *boost* and our workload-aware method *opt*.

As expected, the error of *basic* is exactly proportional to the querying length because a query with longer length usually involves more intervals. However, for the second and third methods, a longer query may hit the nodes on higher levels of the tree index, and thus is affected by much less noise for differential privacy preservation. However, it is well known that such the tree structure brings larger sensitivity, causing worse accuracy of the second and third methods on short queries, which is verified by the results. Tree-based indices show much better results against *basic* on longer queries. Moreover, the technique proposed in this paper consistently outperforms the standard index construction method on almost all lengths. This is because our proposal only slightly adjusts the tree structure, without incurring any additional sensitivity growth. The result structure also better reflects the needs of the queries on these sub-domains, thus improving the hit rate of nodes on higher levels of the index trees.

Figure 12 and Figure 13 validate the effectiveness of tree structure optimization technique compared with *HM*. Here, the fan-out of hierarchical tree is set to the default value 2 and global sensitivity is equal to the maximal length of sequences. The performance of *HM* is not satisfactory, because of lack of optimizations targeting at high sensitivity of sequence database. Besides, the fixed hierarchial strategies also affect the effectiveness of optimizations.

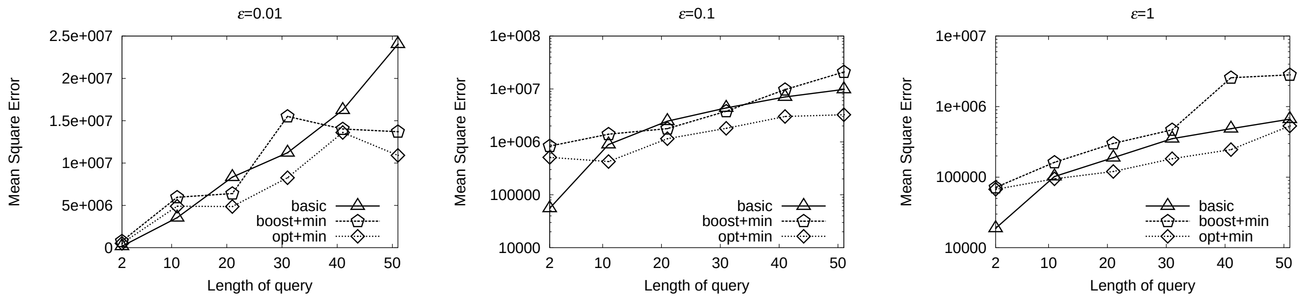


Fig. 11. The effects of tree structure optimization on *San Joaquin*

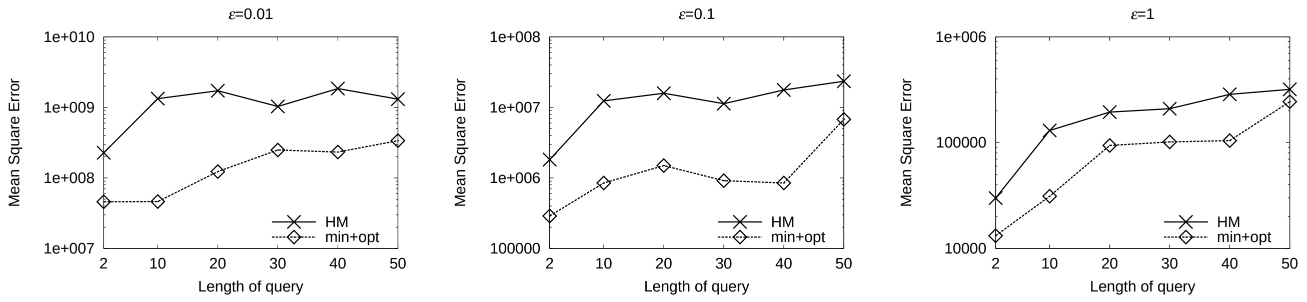


Fig. 12. tree structure optimization vs. hierarchical strategy optimization on *Oldenburg*

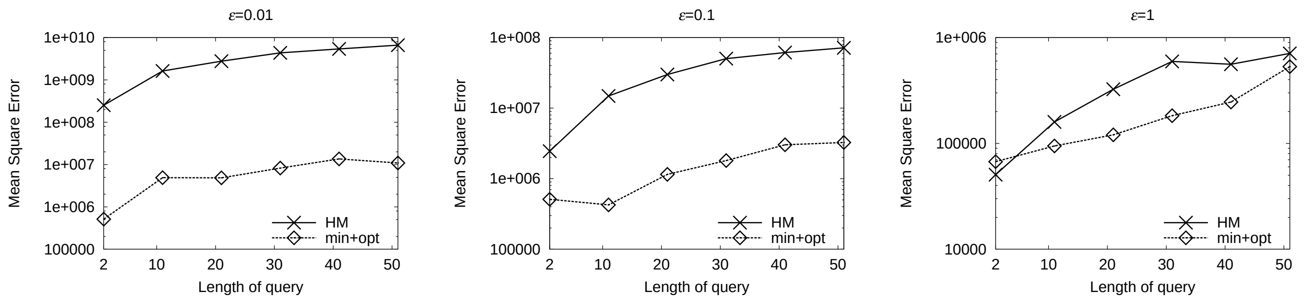


Fig. 13. tree structure optimization vs. hierarchical strategy optimization on *San Joaquin*

## 8 Conclusion

In this paper, a three-step framework is designed for publishing event histograms on privately sensitive sequence data following a graph model. A suite of optimization techniques are proposed to exploit the properties of graph for more accurate event histograms publication with a given sample query workload. And the combination of these techniques dramatically improves the querying performance, as verified over real-world application domains.

In the future, we are going to investigate the possibility of online publication of sequence data, when updates of the sequences are coming into the system on the fly. Moreover, we plan to extend our work to handle the queries containing product operations, which are used for predictive queries under Markov model.