Continual Observation of Joins under Differential Privacy

WEI DONG, Carnegie Mellon University, USA
ZIJUN CHEN and QIYAO LUO, Hong Kong University of Science and Technology, China
ELAINE SHI, Carnegie Mellon University, USA
KE YI, Hong Kong University of Science and Technology, China

The problem of continual observation under differential privacy has been studied extensively in the literature. However, all existing works, with the exception of [28, 50], have only studied the simple counting query and its derivatives. Join queries, which are arguably the most important class of queries in relational databases, have only been considered in [28, 50], but the solutions offered there have two limitations: First, they only support a few specific graph pattern queries, which are special cases of joins. Second, they require hard degree/frequency constraints on the graph/database instance, and the privatized query answers have errors proportional to these constraints.

In this paper, we propose a new differentially private mechanism for continual observation of joins that overcomes these two limitations. Our mechanism supports arbitrary joins and predicates, and do not require any constraints to be given in advance, even over an infinite stream. More importantly, it yields an error that is proportional to the actual maximum degree/frequencies in the graph/database instance at the current time of observation. Such an instance-specific utility guarantee is much preferred for the continual observation problem, where the database size and the query answer may change significantly over time.

CCS Concepts: • **Information systems → Database query processing**; • **Security and privacy → Database and storage security**; • **Theory of computation → Theory of database privacy and security**.

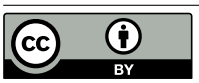
Additional Key Words and Phrases: Differential privacy; Join query; Continual observation

ACM Reference Format:
Wei Dong, Zijun Chen, Qiyao Luo, Elaine Shi, and Ke Yi. 2024. Continual Observation of Joins under Differential Privacy. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 128 (June 2024), 27 pages. https://doi.org/10.1145/3654931

1 INTRODUCTION

Following the influential paper of Dwork et al. [23], a series of works [5, 8–10, 13–15, 17, 24, 28, 30, 31, 41, 47, 49, 53, 55] have extensively studied the problem of *continual observation under differential privacy*. In this problem, we are given a query Q and a possibly infinite stream $\mathbf{I} = (e^{(1)}, e^{(2)}, \dots)$ of tuples arriving over time ($e^{(t)}$ is set to the dummy tuple $\perp$ if no tuple arrives at time t). The goal is to release the query answer $Q(\mathbf{I}^{(t)})$ at each time step $t \in \mathbb{Z}^+$ under *differential privacy (DP)*, where $\mathbf{I}^{(t)} = \{e^{(1)}, \dots, e^{(t)}\}$ is the database instance consisting of all tuples (not including dummy tuples) that have arrived up until time t. Little motivation is needed for this problem, which naturally arises whenever one wishes to monitor private data that evolves over time.

Authors' addresses: Wei Dong, wdong2@cs.cmu.edu, Carnegie Mellon University, Pittsburgh, USA; Zijun Chen, zchendg@cse.ust.hk; Qiyao Luo, qluoak@cse.ust.hk, Hong Kong University of Science and Technology, Hong Kong, China; Elaine Shi, runting@cs.cmu.edu, Carnegie Mellon University, Pittsburgh, USA; Ke Yi, yike@cse.ust.hk, Hong Kong University of Science and Technology, Hong Kong, China.



ACM 2836-6573/2024/6-ART128
https://doi.org/10.1145/3654931

Mechanism		Ours	[28, 50]	Composition
Error level at time t	General join counting queries	$\text{poly}\left(\text{mf}(\mathbf{I}^{(t)})\right)$	Not supported	$\Omega\left(T^{\frac{n}{2}}\right)$
	Triangle counting queries	$\tilde{O}(\widehat{\text{mf}})$	$\Omega(T^{1.5})$	$\tilde{O}\left(\left(\text{mf}(\mathbf{I}^{(t)})\right)^{n-1}\right)$
	n-star counting queries	$\tilde{O}\left(\left(\text{mf}(\mathbf{I}^{(t)})\right)^{n-1}\right)$	$\tilde{O}(\widehat{\text{mf}}^{n-1})$	$\Omega(T^{\frac{n}{2}})$

Table 1. Comparison between our work with prior works. Composition refers to the naïve baseline of using advanced composition on top of known static join-counting schemes [19, 20]. T is the time domain size, n is the number of relations in Q. $\text{mf}(\mathbf{I}^{(t)})$ denotes the maximum frequency of any single attribute in instance $\mathbf{I}$ at time t and $\widehat{\text{mf}}$ is an a-priori upper bound of that. Note that for graph pattern counting queries, $\widehat{\text{mf}} = D$, while $\text{mf}(\mathbf{I}^{(t)})$ is the actual maximum degree of the graph at time t.

However, all past works in the continual setting, with the exception of the recent two [28, 50], have only considered the simple counting query (i.e., $Q(\mathbf{I}) := |\mathbf{I}|$) [10, 23], and its derivatives such as histograms [8, 9, 53], sum queries [5, 15, 33, 47, 55, 56], and linear queries [14, 49]. Most importantly, all these queries have a bounded *global sensitivity* GS_Q, i.e., the query answer (at any particular time) changes by at most GS_Q if the stream contains one more tuple, e.g., $\text{GS}_Q = 1$ for simple counting. On the other hand, join (counting) queries, which are clearly highly useful, have not been thoroughly studied. The main technical challenge is that joins have unbounded global sensitivity. For example, consider a simple two-way join query of the form $Q = |R_1(A) \bowtie R_2(A, B)|$, i.e., we want to join the tables R_1 and R_2 based on the attribute A, and count how many tuples are in the joined table. Each tuple in the input stream is either of the form $R_1\colon (a)$ or $R_2\colon (a, b)$ where the markers R_1 and R_2 denote which table the corresponding tuple belongs to. Suppose initially, the input stream is $\mathbf{I} = \big(R_2\colon (a_1, b_1), R_2\colon (a_1, b_2), \ldots, R_2\colon (a_1, b_T)\big)$, where all tuples belong to R_2. At this moment, there is no join result, i.e., $Q(\mathbf{I}) = 0$. If now a new tuple $R_1\colon (a_1)$ arrives, then $Q(\mathbf{I})$ suddenly becomes T. The amount of change is T, which can be *unbounded* as T goes to infinity.

Fichtenberger et al. [28] and Song et al. [50] have studied the continual observation of joins under differential privacy, among some other graph problems. However, their solution for joins has two limitations: First, their mechanism relies on a query-specific method for calculating the global sensitivity for a given query under the continual setting (see Section 3.4 for more details). They only give the method for triangles and n-stars, which do not work for general graph pattern counting queries. Second, and more importantly, they solve the $\text{GS}_Q = \infty$ issue by introducing a hard, *a priori* degree constraint D on the graph, thus making the global sensitivity bounded. For example, $\text{GS}_Q = D$ for triangle counting, and $\text{GS}_Q = D^{n-1}$ for n-star counting. Consequently, their mechanism has an error proportional to GS_Q for every time step. Additionally, another limitation in their mechanism is that it only supports a finite time domain of a predefined length T. The reason is that they partition the time domain with a tree structure, where the height of the tree must be predetermined so as to allocate the privacy budget to each level. On the other hand, our mechanism supports an infinite time domain by adopting a growing tree structure, and we allocate the privacy budget using a telescoping strategy [10] (see Section 3.4 for more details).

We can generalize their approach to the relational model, where the degree constraints become frequency constraints. GS_Q is thus a function of the these frequency constraints. For example, let $\widehat{\text{mf}}$ denote an a-priori upper bound on the maximum frequency of any single attribute in any table. For simplicity, we will use a single $\widehat{\text{mf}}$ for all tables here, but our technical sections later will give more refined bounds when $\widehat{\text{mf}}$ differs across the tables. Then, for the n-line path join query of the form $Q = \left|R_1(A_1, A_2) \bowtie R_2(A_2, A_3) \bowtie \ldots \bowtie R_n(A_n, A_{n+1})\right|$, GS_Q can be as large as $\widehat{\text{mf}}^{n-1}$. For

a general n-way join query, the global sensitivity depends on the *polymatroid bound* [2] of the boundary queries of the Q. When $n = O(1)$, GS_Q is a polynomial function (denoted $\mathrm{poly}(\widehat{\mathrm{mf}})$) in the frequency constraint $\widehat{\mathrm{mf}}$.

The main drawback of using these hard, *a priori* frequency constraints is that it is difficult to predict the parameters in practice, especially when the stream can be unbounded. If set too conservatively, a large error will be incurred; if set too small, the constraint can be easily violated. Interestingly, we give an example (see Example 5.1) showing that naïvely clipping the database using the given frequency constraints not only hurts accuracy but also violates DP.

Another line of work studies DP single-shot join counting queries for a static database. Although these works [19, 20, 32] can remove the dependency on a-priori frequency constraints, their techniques do not easily generalize to the continual setting. Specifically, running a single-shot mechanism on each time step and naïvely applying the advanced composition theorem [26] results in an error as large as $\Omega(T^{n/2})$, where n is the number of relations involved in the query Q.

Our contributions. This paper proposes the first DP algorithm for general join counting queries under continual observation with infinite time domain. Our algorithm supports arbitrary joins (including self-joins) and predicates, which cover all graph pattern counting queries as special cases, and does not require any constraints on the input stream. More importantly, our algorithm achieves a *time-dependent instance-specific error*: the error at any time step t depends only on the *actual* maximum frequency of any attribute at time step t, henceforth denoted $\mathrm{mf}(\mathbf{I}^{(t)})$. For example, for the n-line path join query, the error at time t is $\tilde{O}\big(\big(\mathrm{mf}(\mathbf{I}^{(t)})\big)^{n-1}\big)$, where the $\tilde{O}$ notation hides ε and polylogarithmic factors. For more general queries where $n = O(1)$, we achieve $\mathrm{poly}\big(\mathrm{mf}(\mathbf{I}^{(t)})\big)$ error at time t. Later in our technical sections, we will give a more refined bound on the error when the actual mf differs across the tables. For specific queries such as the 2-line path counting and n-star counting query, it can be further demonstrated that such an error is optimal [16]. The comparison between our mechanism and prior works are shown in Table 1.

We evaluate the performance of our algorithms using seven real-world graph datasets and the TPC-H benchmark. Our experiments show that for both graph queries and general join queries, achieving such time-dependent instance-specific error significantly improves the accuracy of the algorithm in comparison with previous approaches [28, 50] that make use of *a priori* frequency constraints, even if they are set to be much smaller than T. For example, for 4-star counting query on a graph dataset with $T \approx 10^7$ tuples, we improve the the error by $10^{14}\times$, in comparison with prior approaches [28, 50] where we set the a-priori frequency constraint to $T/1000$.

Technical highlight. In achieving this goal, we have resolved two technical difficulties. First, we design a mechanism to estimate the actual maximum frequency adaptively and in a privacy-preserving fashion as the database grows. In doing so, we can get rid of all *a priori* constraints, even over an infinite stream. Second, since our estimated maximum frequencies can be under-approximating, using them directly to determine how much noise to add may violate the privacy requirement. Therefore, we devise a new clipping mechanism so that the clipped database instance always meets the constraints while preserving differential privacy, which allows for the further application of the mechanism with frequency constraints. Given that our estimated maximum frequencies change over time, a new clipping is invoked at each change. To handle an infinite stream, we allocate the privacy budgets across these clippings in a telescoping manner. We have built such a system prototype to support a wide range of SQL queries and graph pattern counting queries.[1]

[1]Code is available at https://github.com/hkustDB/Dynamic-Join.

Paper organization. The rest of the paper is organized as follows. Section 2 and Section 3 review related work and the necessary preliminaries. In Section 4, we show how to generalize the solution of [28, 50] to arbitrary joins under some given frequency constraints. Section 5 presents our main result: a new clipping mechanism and how to choose the clipping thresholds adaptively so as to achieve an instance-specific utility guarantee. Section 6 and 7 discuss how to handle self-joins, predicates, as well as some implementation details. Experimental results are presented in Section 8 before concluding the paper with some open problems.

2 RELATED WORK

Answering queries in a relational database under DP has been extensively studied in the static setting [3, 6, 16, 18–20, 22, 27, 32, 36, 42, 44, 46, 48, 51, 52]. There are two common DP policies in the relational model: tuple-DP, which protects the privacy of individual tuples, and user-DP, safeguarding users who may possess multiple tuples. Under tuple-DP, the problem can be trivially solved by the Laplace mechanism adding only constant noise if the query does not contain joins. Therefore, significant efforts have been devoted to joins. Most existing works [3, 42, 44, 46, 48] can only handle restricted types of joins. [32] proposed the first mechanism to support arbitrary joins, which are further improved by [19, 20]. Recently, query answering under user-DP has also been studied, where we need additional noise to ensure a higher level privacy protection [6, 16, 18, 22, 27, 36, 52]. As an important special case of join queries, graph pattern counting queries under DP [4, 12, 34, 35, 45, 57] has also been studied. Note that when the edges of the graph are considered as a relation, tuple-DP and user-DP degenerate into edge-DP [4, 34, 45, 57] and node-DP [4, 12, 35], respectively. For more details of answering relational queries under DP in the static setting, see the survey [21].

Under continual observation, tuple-DP corresponds to event-DP, which protects the privacy of the tuple at each time step. Research so far has mostly focused on event-DP and queries without joins. Dwork et al. [23] and Chan et al. [10] initialized the studies by proposing the binary mechanism to answer counting queries and [13, 15, 24, 30, 31, 41] have proposed techniques to reduce the error, either by examining specific instances or by relaxing the privacy definition to (ε, δ)-DP. Besides the simple counting query, other queries such as sum [5, 15, 33, 47, 55, 56], histogram [8, 9, 53], and linear queries [14, 49] have also been explored. For joins, as mentioned earlier, the only existing work is [28, 50], which has studied some specific graph pattern counting queries. They generalize the binary mechanism by assuming some *a priori* degree constraints. A recent work [17] initializes the study of continual observation under user-DP, but it does not consider joins.

3 PRELIMINARIES

3.1 Notation

We first introduce the notation in the static setting. Let $[n] := \{1, \dots, n\}$, and $[i, j] := \{i, \dots, j\}$. Denote $\mathbf{R}$ to be the database schema. In this paper, we primarily focus on multi-way join counting queries in the form of

$$Q := \left|R_1(\mathbf{x}_1) \bowtie \cdots \bowtie R_n(\mathbf{x}_n)\right|,$$

where each $R_i, i \in [n]$ is a relation in $\mathbf{R}$, and $\mathbf{x}_i$ denotes the set of variables/attributes of R_i. For any single variable x, we use $\mathbf{dom}(x)$ to denote the domain of x. For a set of variables $\mathbf{x} = (x_1, \dots, x_k)$, define $\mathbf{dom}(\mathbf{x}) = \mathbf{dom}(x_1) \times \cdots \times \mathbf{dom}(x_k)$.

Let $\mathbf{I}$ be a database instance of $\mathbf{R}$, and $Q(\mathbf{I})$ the result of evaluating Q on $\mathbf{I}$. For each relation $R_i \in \mathbf{R}$, the instance of R_i in $\mathbf{I}$ is denoted by $R_i(\mathbf{I})$. For brevity, we use I_i as a shorthand for $R_i(\mathbf{I})$. Let $N = \sum_i |I_i|$, which is the total instance size. In this paper, we follow the convention of *data complexity* [1], i.e., the complexity is measured in the instance size N while the query size (i.e.,

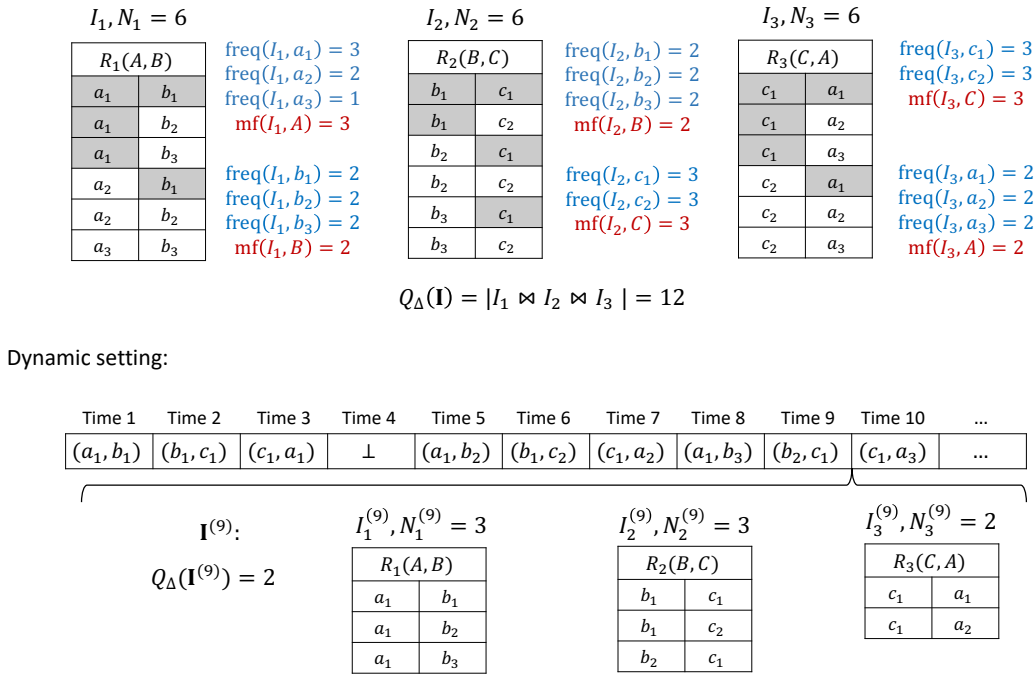


Fig. 1. Example: triangle counting as a joint-counting query $Q_\triangledown(\mathbf{I}) = \left|R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,A)\right|$.

the number of relations and number of variables) is taken as a constant. For any $\mathbf{I}$ and $\mathbf{I}'$, we write $\mathbf{I} \subseteq \mathbf{I}'$ if $I_i \subseteq I_i'$ for every $i \in [n]$.

For any $i \in [n]$, any $\mathbf{x}' \subseteq \mathbf{x}_i$, and any $a \in \mathbf{dom}(\mathbf{x}')$, let $\text{freq}(I_i, a)$ be the number of times a appears in attributes $\mathbf{x}'$ in I_i, i.e.,

$$\text{freq}(I_i, a) = \left|\{e \in I_i \mid \pi_{\mathbf{x}'} e = a\}\right|,$$

and the maximum frequency in attributes $\mathbf{x}'$ in I_i is

$$\text{mf}(I_i, \mathbf{x}') = \max_{a \in \mathbf{dom}(\mathbf{x}')} \text{freq}(I_i, a).$$

Example 3.1. The following is the classical triangle query:

$$Q_\triangledown := \left|R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,A)\right|, \tag{1}$$

which involves three distinct relations and attributes. Figure 1 shows a particular instance and the values of freq and mf on this instance.

For counting the number of triangles in a graph, we can model all the edges as tuples in a single relation $\texttt{Edge}(\texttt{src}, \texttt{dst})$, and rewrite the query using self-joins and variable renaming:

$$\left|\texttt{Edge}(A,B) \bowtie \texttt{Edge}(B,C) \bowtie \texttt{Edge}(C,A)\right|.$$

Note that for counting some other patterns, predicates are needed to exclude degenerate cases, e.g., length-2 paths should be counted by the query

$$\left|\sigma_{A\neq C}\Big(\mathsf{Edge}(A,B)\bowtie\mathsf{Edge}(B,C)\Big)\right|.$$

For most parts of the paper, we focus on queries without self-joins and predicates, which will be discussed in Section 6. □

Under continual observation, which we also call the dynamic setting, the database instance becomes a possibly infinite stream of tuples $\mathbf{I}=\big(R^{(1)}\colon e^{(1)},R^{(2)}\colon e^{(2)},\dots\big)$, where tuple $e^{(t)}$ is added to relation $R^{(t)}$ at t. We set $R^{(t)}$ to be NULL and $x^{(t)}$ to the dummy tuple $\perp$ if no tuple arrives at time t.

For any $t_1\le t_2\in\mathbb{Z}^+$, the database instance within the time interval $[t_1,t_2]$ is denoted as $\mathbf{I}^{[t_1,t_2]}:=\left(I_1^{[t_1,t_2]},\dots,I_n^{[t_1,t_2]}\right)$, where

$$I_i^{[t_1,t_2]}:=\left\{e^{(t)}:t_1\le t\le t_2,R^{(t)}=R_i\right\}.$$

When $t_1=1$, we simplify the notation $[t_1,t_2]$ as (t_2) like $\mathbf{I}^{(t)}=\mathbf{I}^{[1,t]}$, $I_i^{(t)}=I_i^{[1,t]}$, etc. Specially, define $I_i^{(0)}=\emptyset$ for each $i\in[n]$.

In the dynamic setting, we wish to continually monitor the query answers, i.e., the query output also becomes a stream $\mathbf{Q}(\mathbf{I}):=\big(Q(\mathbf{I}^{(1)}),Q(\mathbf{I}^{(2)}),\dots\big)$, and it is required that $Q(\mathbf{I}^{(t)})$ should be outputted at time t. Figure 1 also shows an example in the dynamic setting.

3.2 Differential Privacy

Definition 3.2 (Differential privacy). For any $\varepsilon>0$, a mechanism $\mathcal{M}:\mathcal{I}\to\mathcal{Y}$ is ε-differentially private (DP) if for any neighboring instances $\mathbf{I}\sim\mathbf{I}'\in\mathcal{I}$ and any subset of outputs $Y\subseteq\mathcal{Y}$,

$$\Pr[\mathcal{M}(\mathbf{I})\in Y]\le e^{\varepsilon}\cdot\Pr[\mathcal{M}(\mathbf{I}')\in Y].$$

The exact definition of the neighboring relationship $\sim$ depends on what information is to be protected. In relational databases, two definitions have been adopted in the literature: *tuple-DP* [3, 19, 20, 32, 42, 44, 46, 48] and *user-DP* [6, 16, 18, 27, 36, 52], which respectively generalize edge-DP and node-DP for graph data. In this paper, we adopt the former, while leaving the latter as an interesting open problem. Under tuple-DP, two instances $\mathbf{I}$ and $\mathbf{I}'$ are neighbors if one can be obtained from the other by inserting/deleting one tuple from some relation. More formally, define the distance between two relation instances I_i and I_i' as $d(I_i,I_i'):=|I_i-I_i'|+|I_i'-I_i|$, and for two database instances $\mathbf{I}$ and $\mathbf{I}'$, $d(\mathbf{I},\mathbf{I}'):=\sum_{i\in[n]}d(I_i,I_i')$. In the dynamic setting, we have $d(\mathbf{I},\mathbf{I}'):=\sum_t(|\{e^{(t)}\}-\{e'^{(t)}\}|+|\{e'^{(t)}\}-\{e^{(t)}\}|)$ (set $\{\perp\}=\emptyset$), i.e., it is the minimum number of insertions/deletions needed to convert one stream of tuples to the other. For both scenarios, $\mathbf{I}$ and $\mathbf{I}'$ are neighboring instances if $d(\mathbf{I},\mathbf{I}')=1$.

Note that, in the dynamic setting, the output of $\mathcal{M}$ consists of all the (possibly infinitely many) privatized query answers $\widetilde{Q}(\mathbf{I}^{(t)})$, and they must jointly satisfy Definition 3.2.

The following essential properties of DP will be useful:

Lemma 3.3 (Post Processing [26]). *If $\mathcal{M}:\mathcal{I}\to\mathcal{Y}$ satisfies ε-DP and $\mathcal{M}':\mathcal{Y}\to\mathcal{Z}$ is any randomized mechanism, then $\mathcal{M}'\big(\mathcal{M}(\mathbf{I})\big)$ satisfies ε-DP.*

Lemma 3.4 (Basic Composition Theorem [26]). *If $\mathcal{M}$ is an adaptive composition of differentially private mechanisms $\mathcal{M}_1,\dots,\mathcal{M}_k$, where each $\mathcal{M}_i$ satisfies ε-DP, then $\mathcal{M}$ satisfies $(k\varepsilon)$-DP.*

Algorithm 1: SVT.

Input: $\eta, \varepsilon, f_1(\mathbf{I}), f_2(\mathbf{I}), \ldots$
$\tilde{\eta} \leftarrow \eta + \mathrm{Lap}(2/\varepsilon)$;
for $k \leftarrow 1, 2, \ldots$ **do**
  $\tilde{f_k}(\mathbf{I}) \leftarrow f_k(\mathbf{I}) + \mathrm{Lap}(4/\varepsilon)$;
  if $\tilde{f_k}(\mathbf{I}) > \tilde{\eta}$ **then**
    Break;
  end
end
return k;

Lemma 3.5 (Group Privacy [26]). *If $\mathcal{M}$ is an ε-DP mechanism when neighboring instances are those with distance 1, then $\mathcal{M}$ satisfies $(\lambda\varepsilon)$-DP when neighboring instances are defined as those with distance less than or equal to λ.*

3.3 DP Mechanisms in the Static Setting

In the static setting, the most commonly used DP mechanism is the *Laplace Mechanism*. We describe its d-dimensional version here. Given any query $\mathbf{Q} : \mathcal{I} \to \mathbb{R}^d$, its *global sensitivity* is defined as $\mathrm{GS}_{\mathbf{Q}} = \max_{\mathbf{I} \sim \mathbf{I}'} \left\|\mathbf{Q}(\mathbf{I}) - \mathbf{Q}(\mathbf{I}')\right\|_1$.

Lemma 3.6 (Laplace Mechanism). *Given $\mathbf{Q} : \mathcal{I} \to \mathbb{R}^d$ with global sensitivity $\mathrm{GS}_{\mathbf{Q}}$, the mechanism $\mathcal{M}(\mathbf{I}) = \mathbf{Q}(\mathbf{I}) + \gamma$ preserves ε-DP, where γ is a d-dimensional vector where each entry is independently drawn from the Laplace distribution $\mathrm{Lap}(\mathrm{GS}_{\mathbf{Q}}/\varepsilon)$.*

The utility analysis of the Laplace mechanism makes use of the following concentration property of the Laplace distribution:

Lemma 3.7 ([10]). *Suppose $\gamma_1, \gamma_2, \ldots, \gamma_k$ are independent random variables, where each $\gamma_i \sim \mathrm{Lap}(b_i)$. Then for any $\beta > 0$,*

$$\Pr\left[\left|\sum_i \gamma_i\right| \geq \sqrt{8 \sum_i b_i^2} \cdot \log \frac{2}{\beta}\right] \leq \beta.$$

Another useful tool is the *Sparse Vector Technique (SVT)* [25]. Given a threshold η and a (possibly infinite) sequence of 1-dimensional queries, $f_1, f_2, \ldots$, where each has global sensitivity 1, SVT (described in Algorithm 1) returns the first k such that $f_k(\mathbf{I}) \geq \eta$. Due to the noise, SVT cannot return such an k exactly, but somewhere not too faraway. The formal utility guarantee of SVT is as follows.

Lemma 3.8 ([17]). *Given any $\varepsilon > 0$, SVT satisfies ε-DP and returns a k such that with probability at least $1 - \beta$, $f_k(\mathbf{I}) > \eta - \frac{6}{\varepsilon}\log(2/\beta) - \frac{8}{\varepsilon}\log(k+1)$ and $f_{k'}(\mathbf{I}) < \eta + \frac{8}{\varepsilon}\log(2/\beta) + \frac{6}{\varepsilon}\log(k'+1)$ for all $k' < k$.*

3.4 Binary Mechanism

A basic DP mechanism in the dynamic setting is the *Binary Mechanism (BM)* [10, 23]. Assume for now that the stream has a finite length T, which is a power of 2. The idea is to build a binary hierarchical decomposition of the time domain $[1, T]$ that consists of $\log T + 1$ levels. On level $\ell = 0, 1, \ldots, \log T$, the time domain is partitioned into $T/2^\ell$ canonical intervals, each of length 2^ℓ: $[1, 2^\ell], [2^\ell + 1, 2 \cdot 2^\ell], \ldots, [T - 2^\ell + 1, T]$. Let $\mathcal{T}$ be the set of all such canonical intervals. It is clear that $|\mathcal{T}| = 2T - 1$ and any interval $[1, t]$ is the disjoint union of $\log t$ canonical intervals. We use

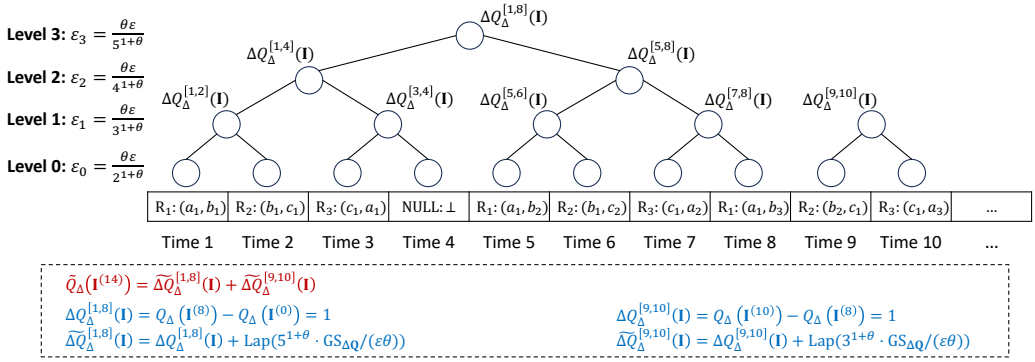


Fig. 2. A running example of the binary mechanism on $Q_\nabla(\mathbf{I}) = \left|R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,A)\right|$.

Algorithm 2: The Binary Mechanism (BM).

Input: $\mathbf{I} = \left(R^{(1)}: e^{(1)}, R^{(2)}: e^{(2)}, \ldots\right)$, ε, $\mathrm{GS}_{\Delta\mathbf{Q}}$, θ

```
for t ← 1, 2, ... do
    foreach canonical interval [t1, t2] such that t2 = t do
        ΔQ̃^[t1,t2](I) ← ΔQ^[t1,t2](I)
        +Lap((log(t2 − t1 + 1) + 2)^{1+θ} · GS_ΔQ/(εθ));
    end
    S ← BinaryForm([1, t]);
    Q̃(I^(t)) ← Σ_{[t1,t2]∈S} ΔQ̃^[t1,t2](I);
end
```

BinaryForm$([1,t])$ to denote this set of canonical intervals that make up $[1,t]$. Thus, any query on $\mathbf{I}^{(t)}$ can be answered by just adding up the query results on these canonical intervals.

Next, we release the privatized results of $Q(\mathbf{I}^{[t_1,t_2]})$ for all the canonical intervals $[t_1,t_2] \in \mathcal{T}$. The observation is that the queries for the canonical intervals on the same level can be regarded as a high-dimensional query. Specifically, the query on level ℓ is

$$\mathbf{Q}^{(\ell)}(\mathbf{I}) := \left(Q(\mathbf{I}^{[1,2^\ell]}), Q(\mathbf{I}^{[2^\ell+1,2\cdot 2^\ell]}), \ldots, Q(\mathbf{I}^{[T-2^\ell+1,T]})\right). \tag{2}$$

For the simple counting query, $\mathbf{Q}^{(\ell)}$ has global sensitivity 1, since adding a tuple at any time step adds 1 to only one canonical interval on each level. Therefore, we can divide the privacy budget by $\log T + 1$ via basic composition and then apply the Laplace mechanism on each level, i.e., masking each $Q(\mathbf{I}^{[t_1,t_2]})$ for $[t_1,t_2] \in \mathcal{T}$ with Laplace noise of scale $(\log T + 1)/\varepsilon$. Then by Lemma 3.7, the total noise for answering any $Q(\mathbf{I}^{(t)})$ is $O(\log^{1.5} T/\varepsilon)$ with constant probability.

The aforementioned idea works for any decomposable query Q, i.e., $Q(\mathbf{I}^{[t_1,t_3]}) = Q(\mathbf{I}^{[t_1,t_2]}) + Q(\mathbf{I}^{[t_2+1,t_3]})$ for any $t_1 \le t_2 < t_3$. However, join queries are not decomposable. To support such queries, the idea to consider *delta queries* [28]: $\Delta Q^{[t_1,t_2]}(\mathbf{I}) := Q(\mathbf{I}^{(t_2)}) - Q(\mathbf{I}^{(t_1-1)})$. This way, any $Q(\mathbf{I}^{(t)})$ is still the sum of $O(\log t)$ delta queries and the binary mechanism still works, except that the high-dimensional query on each level are now formed by the delta queries on this level, i.e., (2) becomes

$$\Delta\mathbf{Q}^{(\ell)}(\mathbf{I}) := \left(\Delta Q^{[1,2^\ell]}(\mathbf{I}), \Delta Q^{[2^\ell+1,2\cdot 2^\ell]}(\mathbf{I}), \ldots, \Delta Q^{[T-2^\ell+1,T]}(\mathbf{I})\right). \tag{3}$$

Since $\Delta\mathbf{Q}^{(\ell)}(\mathbf{I})$ exhibits the highest sensitivity at level 0, the only remaining issue is to bound $\mathrm{GS}_{\Delta\mathbf{Q}^{(0)}}$, the global sensitivity of $\Delta\mathbf{Q}^{(0)}(\mathbf{I})$. For notational simplicity, we just write it as $\mathrm{GS}_{\Delta\mathbf{Q}}$. Note that for the simple counting query, $Q(\mathbf{I}^{[t_1,t_2]})$ and $\Delta Q^{[t_1,t_2]}(\mathbf{I})$ are identical, so $\mathrm{GS}_{\Delta\mathbf{Q}} = 1$. However, this might not be the case for other queries. [28] has derived $\mathrm{GS}_{\Delta\mathbf{Q}}$ for some specific graph pattern counting queries; later, we will prove a more general result that holds for all monotonic and supermodular queries, which include all join and graph pattern counting queries as special cases.

To accommodate an infinite stream, we build $\mathcal{T}$ incrementally, i.e., it only includes all canonical intervals $[t_1, t_2]$ for $t_2 \le t$, where t is the current time. Then instead of allocating the privacy budget equally to all levels, a telescoping strategy [10] is adopted, where the privacy budget for the i-th level is $\varepsilon_i := \frac{\varepsilon\theta}{(i+2)^{1+\theta}}$, where $\theta > 0$ is any small constant. The detailed algorithm is shown in Algorithm 2 and a running example is given in Figure 2.

BM has the following utility guarantee:

Lemma 3.9 ([10]). *Given any $\varepsilon, \theta > 0$, BM satisfies ε-DP, and for any $\beta > 0$ and any $t \in \mathbb{N}$, with probability at least $1 - \beta$, it returns a $\widetilde{Q}(\mathbf{I}_t)$ such that*

$$\left|\widetilde{Q}(\mathbf{I}^{(t)}) - Q(\mathbf{I}^{(t)})\right| = O\Big(\frac{1}{\varepsilon\theta} \cdot \mathrm{GS}_{\Delta\mathbf{Q}} \cdot \log^{1.5+\theta} t \cdot \log(1/\beta)\Big).$$

It is worth mentioning that [10] has proposed another technique to handle an infinite stream without incurring the extra θ term in utility. However, that solution only works for decomposable queries.

Notation	Meaning
$\mathbf{R}$	Database schema
$R_1, \ldots, R_n$	Relation names
$\mathbf{I}, \mathbf{I}'$	Database instances
$e^{(t)}$	Tuple coming at time t
$I_1, \ldots, I_n$	Relation instances
$\mathbf{I}^{(t)}$	Prefix database instance $\mathbf{I}$ of time t
$I_i^{(t)}$	Prefix relation instance I_i of time t
$\mathrm{mf}(I_i, \mathbf{x}')$	Max-frequency of $\mathbf{x}'$ in instance I_i
$\widehat{\mathrm{mf}}(R_i, \mathbf{x}')$	An predefined upper bound of $\widehat{\mathrm{mf}}(I_i, \mathbf{x}')$
$\mathcal{B}_{Q,i}$	Set of boundaries of R_i
m_i	$m_i = \lvert\mathcal{B}_{Q,i}\rvert$
$m_{\max}$	$m_{\max} = \max_i m_i$
$\mathcal{B}_Q$	$\mathcal{B}_Q = \{(i, \mathbf{x}) : \mathbf{x} \in \mathcal{B}_{Q,i}\}$
$\Delta Q^{[t_1,t_2]}(\mathbf{I})$	Change of Q between time interval $[t_1, t_2]$
GS_Q	Global sensitivity of Q
$\mathrm{GS}_{\Delta\mathbf{Q}}$	GS of $\{\Delta Q^{[1,1]}(\mathbf{I}), \Delta Q^{[2,2]}(\mathbf{I}), \ldots\}$

Table 2. Notation used in the paper.

4 THE GLOBAL SENSITIVITY OF JOINS

As seen above, to use the BM on any query Q, it boils down to bounding $\mathrm{GS}_{\Delta\mathbf{Q}}$, which can be further shown to depend on GS_Q, the global sensitivity of Q in the static setting. Below, we first review and clarify existing work in the static setting, and then extend these results to the dynamic setting.

4.1 The Static Setting

As mentioned, the global sensitivity of joins is unbounded. Thus, all prior work has restricted the allowable instances with frequency constraints. Specifically, for every $i \in [n]$ and every subset

of variables $\mathbf{x} \subseteq \mathbf{x}_i$, a frequency upper bound $\widehat{\mathrm{mf}}(R_i, \mathbf{x})$ should be set *a priori*, and all allowable instances $\mathbf{I} = \{I_i\}_{i\in[n]}$ must satisfy $\mathrm{mf}(I_i, \mathbf{x}) \le \widehat{\mathrm{mf}}(R_i, \mathbf{x})$ for all $i, \mathbf{x}$. Then, GS_Q can be bounded in terms of these frequency constraints. For example, [32, 36] have derived the following bound on $\mathrm{GS}_{Q_\triangledown}$ for the triangle query:

$$\mathrm{GS}_{Q_\triangledown} \le \max\left(\widehat{\mathrm{mf}}(R_2, B)\cdot\widehat{\mathrm{mf}}(R_3, A), \widehat{\mathrm{mf}}(R_1, B)\cdot\widehat{\mathrm{mf}}(R_3, C), \widehat{\mathrm{mf}}(R_2, C)\cdot\widehat{\mathrm{mf}}(R_1, A)\right). \tag{4}$$

We observe that this bound is far from tight, and can improve it by borrowing two ideas from the literature. First, [20] shows that GS_Q can be computed from the join size upper bounds of several sub-queries. Consider again the triangle query. Since GS_Q is the maximum amount of change in Q if one tuple is added/removed from R_1, R_2, or R_3. These changes are precisely captured by the following sub-queries:

$$Q_{\triangledown,1} := \Big|(a, b) \bowtie R_2(b, C) \bowtie R_3(C, a)\Big|,$$
$$Q_{\triangledown,2} := \Big|R_1(A, b) \bowtie (b, c) \bowtie R_3(c, A)\Big|,$$
$$Q_{\triangledown,3} := \Big|R_1(a, B) \bowtie R_2(B, c) \bowtie R_3(c, a)\Big|.$$

Subsequently, we have $\mathrm{GS}_{Q_\triangledown} \le \max_{\mathbf{I}} \max\left\{Q_{\triangledown,1}(\mathbf{I}), Q_{\triangledown,2}(\mathbf{I}), Q_{\triangledown,3}(\mathbf{I})\right\}$.

More generally, given a query Q, for each relation R_i we define $\mathcal{B}_{Q,i}$ as the *boundary* of R_i, which comprises of the variables that R_i shares with another relation R_j for $j \in [n]$, i.e.,

$$\mathcal{B}_{Q,i} := \{\mathbf{x}_i \cap \mathbf{x}_j : j \in [n], j \ne i, \mathbf{x}_i \cap \mathbf{x}_j \ne \emptyset\}.$$

Let $m_i = |\mathcal{B}_{Q,i}|$, $m = \sum_i m_i$, and $m_{\max} = \max_i m_i$. Furthermore, define

$$\mathcal{B}_Q := \left\{(i, \mathbf{x}) : i \in [n], \mathbf{x} \in \mathcal{B}_{Q,i}\right\}.$$

For example, on the triangle query, we have $\mathcal{B}_{Q,1} = \left\{\{A\}, \{B\}\right\}$, $\mathcal{B}_{Q,2} = \left\{\{B\}, \{C\}\right\}$, $\mathcal{B}_{Q,3} = \left\{\{C\}, \{A\}\right\}$, $\mathcal{B}_Q = \left\{\big(1, \{A\}\big), \big(1, \{B\}\big), \big(2, \{B\}\big), \big(2, \{C\}\big), \big(3, \{C\}\big), \big(3, \{A\}\big)\right\}$, and $m_1 = m_2 = m_3 = 2, m = 6, m_{\max} = 2$.

Let Q_i be the sub-query of Q where all variables in $\mathcal{B}_{Q,i}$ are set to constants (like $Q_{\triangledown,1}$, $Q_{\triangledown,2}$, and $Q_{\triangledown,3}$ shown above). Then we can bound GS_Q as

$$\mathrm{GS}_Q \le \max_{i\in[n]} \max_{\mathbf{I}} Q_i(\mathbf{I}). \tag{5}$$

It now remains to bound each $\max_{\mathbf{I}} Q_i(\mathbf{I})$ under the given frequency constraints $\left\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\right\}_{(i,\mathbf{x})\in\mathcal{B}_Q}$. We observe that this is precisely the problem studied in [2, 29], for which the best efficiently computable join size upper bound is the *polymatroid bound*. Plugging the polymatroid bound into (5) then yields a bound on GS_Q, which is a function of $\left\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\right\}_{(i,\mathbf{x})\in\mathcal{B}_Q}$. The polymatroid bound in its full generality is complicated; instead, we have derived $\mathrm{GS}_Q\left(\left\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\right\}_{(i,\mathbf{x})\in\mathcal{B}_Q}\right)$ for several common queries in Table 3. These bounds are much tighter than the previous bounds [32, 36]. For example, for the triangle query and suppose all the frequency constraints are equal, then the bound in Table 3 is quadratically smaller than the previous bound in (4). In the sequel, we often omit the subscript $(i, \mathbf{x}) \in \mathcal{B}_Q$ for notational simplicity.

4.2 The Dynamic Setting

Moving forward to the dynamic setting, we need to bound the global sensitivity of the delta queries $\Delta\mathbf{Q}$. Earlier work [28] has considered this problem for several specific graph pattern counting

Query	$\mathrm{GS}_Q\left(\left\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\right\}_{(i,\mathbf{x})\in\mathcal{B}_Q}\right)$
$\left\lvert R_1(A,B) \bowtie R_2(B,C)\right\rvert$	$\max\left(\widehat{\mathrm{mf}}(R_1,B), \widehat{\mathrm{mf}}(R_2,B)\right)$
$\left\lvert R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D)\right\rvert$	$\max\left(\widehat{\mathrm{mf}}(R_2,B)\cdot\widehat{\mathrm{mf}}(R_3,C), \widehat{\mathrm{mf}}(R_1,B)\cdot\widehat{\mathrm{mf}}(R_3,C), \widehat{\mathrm{mf}}(R_1,B)\cdot\widehat{\mathrm{mf}}(R_2,C)\right)$
$\left\lvert R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,A)\right\rvert$	$\max\Big(\min\left(\widehat{\mathrm{mf}}(R_2,B), \widehat{\mathrm{mf}}(R_3,A)\right), \min\left(\widehat{\mathrm{mf}}(R_1,B), \widehat{\mathrm{mf}}(R_3,C)\right), \min\left(\widehat{\mathrm{mf}}(R_2,C), \widehat{\mathrm{mf}}(R_1,A)\right)\Big)$
$\left\lvert R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D) \bowtie R_4(D,A)\right\rvert$	$\max\Big(\min\left(\widehat{\mathrm{mf}}(R_2,B)\cdot\widehat{\mathrm{mf}}(R_3,C), \widehat{\mathrm{mf}}(R_3,D)\cdot\widehat{\mathrm{mf}}(R_4,A), \widehat{\mathrm{mf}}(R_2,B)\cdot\widehat{\mathrm{mf}}(R_4,A)\right), \min\left(\widehat{\mathrm{mf}}(R_3,C)\cdot\widehat{\mathrm{mf}}(R_4,D), \widehat{\mathrm{mf}}(R_1,B)\cdot\widehat{\mathrm{mf}}(R_4,A), \widehat{\mathrm{mf}}(R_3,C)\cdot\widehat{\mathrm{mf}}(R_1,B)\right), \min\left(\widehat{\mathrm{mf}}(R_4,D)\cdot\widehat{\mathrm{mf}}(R_1,A), \widehat{\mathrm{mf}}(R_2,C)\cdot\widehat{\mathrm{mf}}(R_1,B), \widehat{\mathrm{mf}}(R_4,D)\cdot\widehat{\mathrm{mf}}(R_2,C)\right), \min\left(\widehat{\mathrm{mf}}(R_1,A)\cdot\widehat{\mathrm{mf}}(R_2,C), \widehat{\mathrm{mf}}(R_3,D)\cdot\widehat{\mathrm{mf}}(R_2,C), \widehat{\mathrm{mf}}(R_3,D)\cdot\widehat{\mathrm{mf}}(R_1,A)\right)\Big)$
$\left\lvert R_1(A,B) \bowtie R_2(A,C) \bowtie R_3(A,D)\right\rvert$	$\max\left(\widehat{\mathrm{mf}}(R_2,A)\cdot\widehat{\mathrm{mf}}(R_3,A), \widehat{\mathrm{mf}}(R_1,A)\cdot\widehat{\mathrm{mf}}(R_3,A), \widehat{\mathrm{mf}}(R_1,A)\cdot\widehat{\mathrm{mf}}(R_2,A)\right)$

Table 3. Formulating $\mathrm{GS}_Q\left(\left\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\right\}_{(i,\mathbf{x})\in\mathcal{B}_Q}\right)$ with polymatroid bound for common join counting queries.

queries. Below we prove a more general result that $\mathrm{GS}_{\Delta Q} \le \mathrm{GS}_Q$ as long as Q is *monotonic* and *supermodular*, i.e., for any $\mathbf{I}_1 \subseteq \mathbf{I}_2$, and any $\mathbf{I}_3$,

$$Q(\mathbf{I}_1) \le Q(\mathbf{I}_1 \cup \mathbf{I}_3), \tag{Monotonic}$$

$$Q(\mathbf{I}_1 \cup \mathbf{I}_3) - Q(\mathbf{I}_1) \le Q(\mathbf{I}_2 \cup \mathbf{I}_3) - Q(\mathbf{I}_2). \tag{Supermodular}$$

Lemma 4.1. *For any monotonic and supermodular Q, we have $\mathrm{GS}_{\Delta Q} \le \mathrm{GS}_Q$.*

Proof. Recall that $\mathrm{GS}_{\Delta Q}$ is the global sensitivity of the following T-dimensional query at level 0 (T can go to infinite):

$$\Delta\mathbf{Q}^{(0)}(\mathbf{I}) := \left(\Delta Q^{[1,1]}(\mathbf{I}), \Delta Q^{[2,2]}(\mathbf{I}), \ldots, \Delta Q^{[T,T]}(\mathbf{I})\right).$$

Let $\mathbf{I} \sim \mathbf{I}'$ be any two neighboring input streams. Without loss of generality, assume that only difference between $\mathbf{I}$ and $\mathbf{I}'$ happens at time t' where $e^{(t')} \neq \perp$ while $e'^{(t')} = \perp$.

Initially, both instances $\mathbf{I}$ and $\mathbf{I}'$ are empty, thus

$$Q(\mathbf{I}^{(0)}) = Q(\mathbf{I}'^{(0)}). \tag{6}$$

By the definition GS_Q, we have

$$\left|Q(\mathbf{I}^{(T)}) - Q(\mathbf{I}'^{(T)})\right| \le \mathrm{GS}_Q. \tag{7}$$

Recalling $Q(\mathbf{I}) = Q(\mathbf{I}^{(0)}) + \sum_t \Delta Q^{[t,t]}(\mathbf{I})$ and $Q(\mathbf{I}') = Q(\mathbf{I}'^{(0)}) + \sum_t \Delta Q^{[t,t]}(\mathbf{I}')$ and integrating these with (6) and (7), we have

$$\left|\sum_t \left(\Delta Q^{[t,t]}(\mathbf{I}) - \Delta Q^{[t,t]}(\mathbf{I}')\right)\right| \le \mathrm{GS}_Q. \tag{8}$$

For any $t < t'$, it is clear that $\Delta Q^{[t,t]}(\mathbf{I}') = \Delta Q^{[t,t]}(\mathbf{I})$ and by invoking the monotonic property of Q, we have

$$\Delta Q^{[t',t']}(\mathbf{I}) \ge 0 = \Delta Q^{[t',t']}(\mathbf{I}').$$

Moreover, for any $t > t'$, the condition $\mathbf{I'}^{(t-1)} \subseteq \mathbf{I}^{(t-1)}$ holds true and $e^{(t)} = e'^{(t)}$ leading to

$$\begin{aligned}\Delta Q^{[t,t]}(\mathbf{I}) =& Q\big(\mathbf{I}^{(t-1)} \cup \{e^{(t)}\}\big) - Q(\mathbf{I}^{(t-1)}) \\ =& Q\big(\mathbf{I}^{(t-1)} \cup \{e'^{(t)}\}\big) - Q(\mathbf{I}^{(t-1)}) \\ \geq& Q\big(\mathbf{I'}^{(t-1)} \cup \{e'^{(t)}\}\big) - Q(\mathbf{I'}^{(t-1)}) \\ =& \Delta Q^{[t,t]}(\mathbf{I'}),\end{aligned} \tag{9}$$

where the inequality in the third line is by the supermodular property of Q.

Combining (8) and (9), we have

$$\sum_t \left|\Delta Q^{[t,t]}(\mathbf{I}) - \Delta Q^{[t,t]}(\mathbf{I'})\right| \leq \mathrm{GS}_Q,$$

as desired. □

Note that any multi-way join query is both monotonic and supermodular. Monotonicity is trivial. To see supermodularity, consider adding a tuple e to $\mathbf{I}$. The join size will increase by $|e \bowtie \mathbf{I}|$. When $\mathbf{I}$ has more tuples, $|e \bowtie \mathbf{I}|$ cannot be less. Thus, the BM and Lemma 3.9 immediately generalize to any multi-way join query, with an error proportional to $\mathrm{GS}_Q\Big(\big\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\big\}\Big)$. We note that [28] has also derived $\mathrm{GS}_{\Delta Q}$ for triangle counting query and n-star counting queries. Our general polymatroid bound $\mathrm{GS}_Q\Big(\big\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\big\}\Big)$ degenerates into their bounds on these two specific queries.

5 DYNAMIC CLIPPING MECHANISM

Although Lemma 4.1 has enabled the binary mechanism to work for any multi-way join query, it is far from satisfactory. First, it requires the frequency constraints $\big\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\big\}$ to be set *a priori*. This is problematic, especially for an unbounded stream. In practice, it is also not easy to give a reasonable constraint for every $(i, \mathbf{x}) \in \mathcal{B}_Q$: The triangle query already requires 6 constraints, other queries may require more. Second and more importantly, its error at every time step is proportional to $\mathrm{GS}_Q\Big(\big\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\big\}\Big)$, even when the current database is small and nowhere near the constraints. Ideally, the error should be proportional to $\mathrm{GS}_Q\Big(\big\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\big\}\Big)$, i.e., it only depends on the actual maximum frequencies of the database at the current time t. In this section, we present the main result of this paper, a dynamic clipping mechanism that achieves this instance-specific error without requiring any frequency constraints given in advance.

In the static setting, a prevalent technique for achieving an instance-specific error is the *clipping* (or *truncation*) mechanism [16, 32, 36]. For a clipping threshold τ, all tuples with influence more than τ are clipped. Then, the global sensitivity of the query is limited by τ (or some function of τ) and the Laplace mechanism is then invoked.

However, a straightforward extension of the clipping mechanism to the dynamic setting will not work. Let $\bar{\mathbf{I}}$ denote the clipped instance, and $\bar{\mathbf{I}}^{(t)}$ and $\bar{I}_i^{(t)}$ are prefix database instance and prefix relation instance of time t. Suppose we use a clipping threshold $\tau(R_i, \mathbf{x})$ for each $(i, \mathbf{x}) \in \mathcal{B}_Q$, and clip an incoming tuple $e^{(t)}$ if $\mathrm{freq}\Big(\bar{I}_i^{(t-1)}, \pi_{\mathbf{x}} e^{(t)}\Big) = \tau(R_i, \mathbf{x})$ for some $(i, \mathbf{x}) \in \mathcal{B}_Q$. The clipped database instance will thus satisfy the frequency constraints $\big\{\mathrm{freq}(\bar{I}_i^{(t)}, \mathbf{x}) \leq \tau(R_i, \mathbf{x})\big\}$ for any times $t \in \mathbb{Z}^+$, and it may appear that we can then use the BM with $\mathrm{GS}_{\Delta Q} = \mathrm{GS}_Q\Big(\big\{\tau(R_i, \mathbf{x})\big\}\Big)$. However, the following example shows that this breaks privacy.

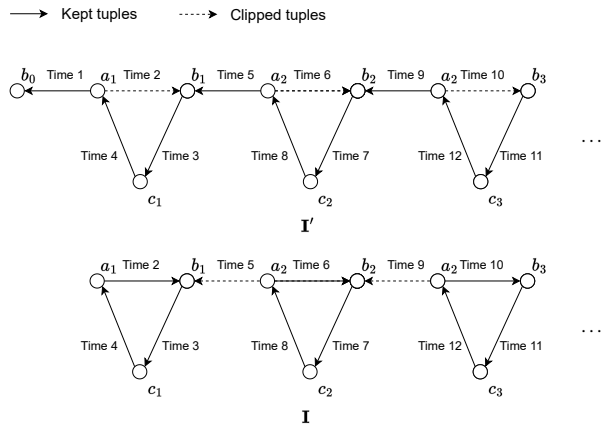


Fig. 3. An example showing that the naïvely clipping mechanism breaks privacy.

Example 5.1. Consider the triangle query $Q_\triangledown = |R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,A)|$, where $\mathbf{dom}(A) = \{a_1, a_2, \dots, \}, \mathbf{dom}(B) = \{b_0, b_1, b_2, \dots, \}, \mathbf{dom}(C) = \{c_1, c_2, \dots\}$. The two neighboring instances $\mathbf{I}$ and $\mathbf{I}'$ are shown in Figure 3, where $\mathbf{I}$ has one less tuple at time 1: $e^{(1)} = \perp$. Suppose $\tau(R_i, \mathbf{x}) = 1$ for all $(i, \mathbf{x}) \in \mathcal{B}_Q$, i.e., all the in-degrees and out-degrees are constrained to 1. Then we see that on the clipped instances, at every time $t \in \mathbb{Z}^+$, we have $Q_\triangledown(\bar{\mathbf{I}}^{(t)}) = \lfloor t/4 \rfloor$ while $Q_\triangledown(\bar{\mathbf{I}}^{'(t)}) = 0$. On the other hand, $\mathrm{GS}_Q\Big(\big\{\tau(R_i, \mathbf{x}) = 1\big\}\Big) = 1$, but adding a noise of scale 1 cannot mask this difference. □

Fundamentally, the key condition needed by the BM is that $\mathrm{GS}_Q\Big(\big\{\tau(R_i, \mathbf{x})\big\}\Big)$ is the maximum difference in the query result between any two neighboring instances both satisfying the frequency constraints. However, while the clipped instance satisfies the frequency constraints after clipping, two neighboring instances (before clipping) may not be neighbors anymore after the clipping, as illustrated in the example above.

In the dynamic setting, another challenge is that we must dynamically select a clipping threshold as the instance grows over time in order to achieve an error that depends on the current instance, as opposed to the static setting where the threshold is only computed once and for all. Note that the growing clipping threshold must be continuously selected in a differentially private fashion. In the next two subsections, we show how to overcome these two challenges.

5.1 A Private Clipping Mechanism

In this subsection, we show how to clip a stream of tuples under a set of fixed clipping threshold $\big\{\tau(R_i, \mathbf{x})\big\}$, so that two neighboring streams are still k-neighbors after the clipping with k equal to some constant. Then it would be safe to feed the clipped streams to the BM while satisfying DP.

The idea is, in addition to the clipped database instance, we also maintain the unclipped database. When deciding if a tuple $e^{(t)}$ should be clipped, we check its frequencies in the unclipped database: If $\mathrm{freq}\Big(I_i^{(t-1)}, \pi_{\mathbf{x}} e^{(t)}\Big) \geq \tau(R_i, \mathbf{x})$ for some $(i, \mathbf{x}) \in \mathcal{B}_Q$ in the unclipped database, e will be clipped. Note that the condition uses $\geq$ instead of $=$ as in the naïvely clipping mechanism, since the frequencies may exceed the clipping threshold in the unclipped database. Applying our new clipping mechanism on Example 5.1, the clipped streams are shown in Figure 4, which differ in 2 time steps (time 1 and

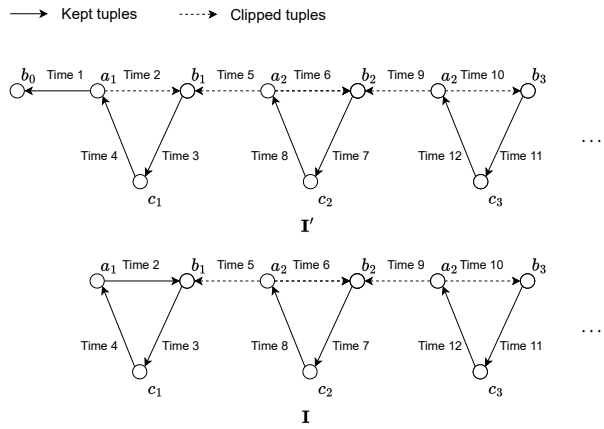


Fig. 4. An example of the new clipping mechanism.

2), i.e., they are still distance-2 neighbors. This means that they can still be fed to BM, except with a privacy budget of $\varepsilon/2$ by the group privacy property of DP (Lemma 3.5).

It turns out that the general guarantee we can prove below is just slightly worse than this example (recall that $m_{\max} = 2$ for the triangle query). Here, we use $\mathrm{Clip}\Big(\mathbf{I}, \big\{\tau(R_i, \mathbf{x})\big\}\Big)$ to denote the stream clipped with thresholds $\big\{\tau(R_i, \mathbf{x})\big\}$. More formally, $\mathrm{Clip}\Big(\mathbf{I}, \big\{\tau(R_i, \mathbf{x})\big\}\Big)$ $= \big(\overline{R}^{(1)} : \overline{e}^{(1)}, \overline{R}^{(2)} : \overline{e}^{(2)}, \ldots\big)$, where for each $t \in \mathbb{Z}^+$, let $R^{(t)} = R_{i'}$, then

$$\overline{R}^{(t)}, \overline{e}^{(t)} = \begin{cases} R^{(t)}, e^{(t)} & \forall \mathbf{x} \in \mathcal{B}_{Q,i'}, \mathrm{freq}\Big(I_{i'}^{(t-1)}, \pi_{\mathbf{x}} e^{(t-1)}\Big) < \tau(R_{i'}, \mathbf{x}) \\ \text{NULL}, \perp & \text{Otherwise.} \end{cases}$$

Lemma 5.2. *Given any* $\mathbf{I} \sim \mathbf{I}'$ *and any* $\big\{\tau(R_i, \mathbf{x})\big\}$,

$$d\left(\mathrm{Clip}\Big(\mathbf{I}, \big\{\tau(R_i, \mathbf{x})\big\}\Big), \mathrm{Clip}\Big(\mathbf{I}', \big\{\tau(R_i, \mathbf{x})\big\}\Big)\right) \le m_{\max} + 1.$$

Proof. Let us consider two instances $\mathbf{I}$ and $\mathbf{I}'$ such that $\mathbf{I}' \subseteq \mathbf{I}$, with $\mathbf{I}$ and $\mathbf{I}'$ differing by $e^{(t')}$ with $R^{(t')} = R_{i'}$, i.e., $e^{(t')} \in I_{i'}^{(t')}$, $e'^{(t')} = \perp$.

Given that $\mathbf{I}' \subseteq \mathbf{I}$, it follows that for any time $t \in \mathbb{Z}^+$, any $i \in [n]$, any $\mathbf{x} \subseteq \mathbf{x}_i$, and any $a \in \mathbf{dom}(\mathbf{x})$, we always have $\mathrm{freq}(I'^{(t)}_i, a) \le \mathrm{freq}(I_i^{(t)}, a)$. As a consequence, excluding the tuple $e^{(t')}$, the clipped instance of $\mathbf{I}$ will not contain any additional tuples compared to the clipped instance of $\mathbf{I}'$, i.e.,

$$\left|\mathrm{Clip}\Big(\mathbf{I}, \big\{\tau(R_i, \mathbf{x})\big\}\Big) - \mathrm{Clip}\Big(\mathbf{I}', \big\{\tau(R_i, \mathbf{x})\big\}\Big)\right| \le 1. \tag{10}$$

Furthermore, at any given time $t \in \mathbb{Z}^+$, the tuple $e^{(t')}$ will only affect $\mathrm{freq}\Big(I_{i'}^{(t)}, \pi_{\mathbf{x}} e^{(t')}\Big)$ for $\mathbf{x} \in \mathcal{B}_{Q,i'}$, each of which can at most result in one additional tuple being clipped in $\mathbf{I}$. Recalling the definition that $m_i = |\mathcal{B}_{Q,i}|$, we have

$$\left|\mathrm{Clip}\Big(\mathbf{I}', \big\{\tau(R_i, \mathbf{x})\big\}\Big) - \mathrm{Clip}\Big(\mathbf{I}, \big\{\tau(R_i, \mathbf{x})\big\}\Big)\right| \le m_i. \tag{11}$$

Combining (10) and (11), we are able to deduce the conclusion. □

Algorithm 3: ClipDP.

Input: $\mathbf{I} = \left(R^{(1)}\colon e^{(1)}, R^{(2)}\colon e^{(2)}, \dots\right)$, ε, θ, $\left\{\tau(R_i, \mathbf{x})\right\}$

Compute $\mathrm{GS}_Q\left(\left\{\tau(R_i, \mathbf{x})\right\}\right)$ as Section 4.1;

Initialize $\mathrm{BM}\left(\mathrm{Clip}\left(\mathbf{I}, \left\{\tau(R_i, \mathbf{x})\right\}\right), \frac{\varepsilon}{m_{\max+1}}, \mathrm{GS}_Q\left(\left\{\tau(R_i, \mathbf{x})\right\}\right)\right)$ with $\mathrm{Clip}\left(\mathbf{I}, \left\{\tau(R_i, \mathbf{x})\right\}\right)$ updated time to time;

for $t \leftarrow 1, 2, \dots$ **do**

  Update $\mathrm{Clip}\left(\mathbf{I}, \left\{\tau(R_i, \mathbf{x})\right\}\right)$ used in BM;

  Answer $\widetilde{Q}(\mathbf{I}^{(t)})$ with BM;

end

The immediate consequence of this lemma is that, to satisfy ε-DP, it is sufficient to feed the clipped stream to BM with a privacy budget of $\varepsilon/(m_{\max}+1)$. We denote this mechanism as ClipDP and the details are shown in Algorithm 3. Next, we analyze the utility. The error of ClipDP consists of two parts: the noise introduced by BM and the bias due the clipping. Feeding the facts that $m_{\max}$ is some small constant, and the clipped stream satisfies the frequency constraints $\left\{\tau(R_i, \mathbf{x})\right\}$, into Lemma 3.9, we can get the noise term bounded by $O\left(\frac{1}{\varepsilon\theta} \cdot \mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right) \cdot \log^{1.5+\theta} t \cdot \log(1/\beta)\right)$. We bound the bias in terms of the number of tuples clipped:

$$\mathrm{ClipNum}\left(\mathbf{I}^{(t)}, \left\{\tau(R_i, \mathbf{x})\right\}\right) = \left|\mathbf{I}^{(t)} - \mathrm{Clip}\left(\mathbf{I}^{(t)}, \left\{\tau(R_i, \mathbf{x})\right\}\right)\right|.$$

Since each clipped tuple can contribute at most $\mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right)$ number of join results, we obtain the following utility guarantee:

Theorem 5.3. *Given any $\varepsilon > 0$, $\theta > 0$, and any clipping thresholds $\left\{\tau(R_i, \mathbf{x})\right\}$, the mechanism* $\mathrm{ClipDP}(\mathbf{I}, \varepsilon, \theta, \{\tau(R_i, \mathbf{x})\})$ *satisfies ε-DP. For any $t \in \mathbb{Z}^+$, with probability at least $1-\beta$, it returns a $\widetilde{Q}(\mathbf{I}^{(t)})$ such that*

$$\left|\widetilde{Q}(\mathbf{I}^{(t)}) - Q(\mathbf{I}^{(t)})\right| \le O\left(\frac{1}{\varepsilon\theta} \cdot \mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right) \cdot \log^{1.5+\theta} t \cdot \log(1/\beta)\right) + \mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right) \cdot \mathrm{ClipNum}\left(\mathbf{I}^{(t)}, \left\{\tau(R_i, \mathbf{x})\right\}\right).$$

5.2 Adaptive Clipping Thresholds

Both the error terms in Theorem 5.3 crucially depend on the clipping thresholds $\left\{\tau(R_i, \mathbf{x})\right\}$. Larger values of $\left\{\tau(R_i, \mathbf{x})\right\}$ will reduce the bias but increase the noise. The optimal choice of thresholds is $\left\{\tau(R_i, \mathbf{x}) = \mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}$ for each time t, which will make the noise term $\tilde{O}\left(\mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right)\right)$, as desired, and the second term 0. However, using these thresholds directly violates DP as they depend on the actual instance.

To find privacy-preserving and near-optimal clipping thresholds, the idea is to start with a small threshold $\tau(R_i, \mathbf{x}) = 2$ at the beginning. After excessive tuples have been clipped, we double the value of $\tau(R_i, \mathbf{x})$ and (conceptually) rerun ClipDP with the new thresholds. To determine the right moment to double $\tau(R_i, \mathbf{x})$ in a privacy-preserving manner, it is crucial first to establish a method

Algorithm 4: DynamicClipDP.

Input: $\mathbf{I} = \left(R^{(1)}: e^{(1)}, R^{(2)}: e^{(2)}, \dots\right)$, ε, β, θ

$k_{\mathrm{C}} \leftarrow 1$; // Initialize the parameters for the first ClipDP

$\varepsilon_{\mathrm{C}} \leftarrow \varepsilon\theta/2^{2+\theta}$;

Let $\tau(R_i, \mathbf{x}) \leftarrow 2$ for each $(i, \mathbf{x}) \in \mathcal{B}_Q$;

Start ClipDP$\left(\mathbf{I}, \varepsilon_{\mathrm{C}}, \theta, \left\{\tau(R_i, \mathbf{x})\right\}\right)$;

for $(i, \mathbf{x}) \in \mathcal{B}_Q$ **do**

  $k_{\mathrm{S}_{(i,\mathbf{x})}} \leftarrow 1$; // Initialize the parameters for the first SVT associated with $\tau(R_i, \mathbf{x})$

  $\varepsilon_{\mathrm{S}_{(i,\mathbf{x})}} \leftarrow \varepsilon\theta/(m_{\max} \cdot 2^{2+\theta})$, $\beta_{\mathrm{S}_{(i,\mathbf{x})}} \leftarrow \beta/(m \cdot 2^3)$;

  Initialize $\mathrm{SVT}_{(i,\mathbf{x})} \leftarrow \mathrm{SVT}\left(0, \varepsilon_{\mathrm{SVT}_{(i,\mathbf{x})}}, f_1(\mathbf{I}), f_2(\mathbf{I}), \dots\right)$ with $f_t(\mathbf{I})$'s inputted later;

end

for $t \leftarrow 1, 2, \dots$ **do**

  do

    SVTStop $\leftarrow$ False; // Check whether any $\tau(R_i, \mathbf{x})$ needs to be doubled

    for $(i, \mathbf{x}) \in \mathcal{B}_Q$ **do**

      Feed $f_t(\mathbf{I}) \leftarrow \mathrm{ClipNum}\left(I_i^{(t)}, \tau(R_i, \mathbf{x})\right) - \frac{8}{\varepsilon_{\mathrm{S}_{(i,\mathbf{x})}}} \log(2/\beta_{\mathrm{S}_{(i,\mathbf{x})}}) - \frac{6}{\varepsilon_{\mathrm{S}_{(i,\mathbf{x})}}} \log(t+1)$ into $\mathrm{SVT}_{(i,\mathbf{x})}$;

      if $\mathrm{SVT}_{(i,\mathbf{x})}$ *stops at* t **then**

        $\tau(R_i, \mathbf{x}) \leftarrow \tau(R_i, \mathbf{x}) \cdot 2$; // Double the $\tau(R_i, \mathbf{x})$ and re-run the $k_{\mathrm{S}_{(i,\mathbf{x})}} + 1$th SVT associated with $\tau(R_i, \mathbf{x})$

        $k_{\mathrm{S}_{(i,\mathbf{x})}} \leftarrow k_{\mathrm{S}_{(i,\mathbf{x})}} + 1$;

        $\varepsilon_{\mathrm{S}_{(i,\mathbf{x})}} \leftarrow \varepsilon\theta/\left(2m_{\max} \cdot (k_{\mathrm{S}_{(i,\mathbf{x})}} + 1)^{1+\theta}\right)$, $\beta_{\mathrm{S}_{(i,\mathbf{x})}} \leftarrow \beta/\left(2m \cdot (k_{\mathrm{S}_{(i,\mathbf{x})}} + 1)^2\right)$;

        Re-run $\mathrm{SVT}_{(i,\mathbf{x})} \leftarrow \mathrm{SVT}\left(0, \varepsilon_{\mathrm{S}_{(i,\mathbf{x})}}, f_t(\mathbf{I}), f_{t+1}(\mathbf{I}), \dots\right)$;

        SVTStop $\leftarrow$ True;

        break;

    end

    if SVTStop = True **then**

      $k_{\mathrm{C}} \leftarrow k_{\mathrm{C}} + 1$; // Re-run the ClipDP with updated $\left\{\tau(R_i, \mathbf{x})\right\}_{(i,\mathbf{x}) \in \mathcal{B}_Q}$

      $\varepsilon_{\mathrm{C}} \leftarrow \varepsilon\theta/\left(2 \cdot (k_{\mathrm{C}} + 1)^{1+\theta}\right)$;

      Start ClipDP$\left(\mathbf{I}, \varepsilon_{\mathrm{C}}, \theta, \left\{\tau(R_i, \mathbf{x})\right\}\right)$;

  while SVTStop = True;

  Use ClipDP to answer $\widetilde{Q}(\mathbf{I}^{(t)})$;

end

for quantifying the number of tuples clipped by the current $\tau(R_i, \mathbf{x})$. For any $(i, \mathbf{x}) \in \mathcal{B}_Q$, let

$$\mathrm{ClipNum}\left(I_i^{(t)}, \tau(R_i, \mathbf{x})\right) = \left|\left\{e \in I_i^{(t)} : \mathrm{freq}\left(I_i^{(t)}, \pi_{\mathbf{x}} e\right) > \tau(R_i, \mathbf{x})\right\}\right|$$

be the number of tuples in $I_i^{(t)}$ that have been clipped due to $\tau(R_i, \mathbf{x})$. Because a tuple may exceed multiple clipping thresholds, the total number of clipped tuples is bounded by their sum:

$$\mathrm{ClipNum}\left(\mathbf{I}^{(t)}, \left\{\tau(R_i, \mathbf{x})\right\}\right) \le \sum_{(i,\mathbf{x}) \in \mathcal{B}_Q} \mathrm{ClipNum}\left(I_i^{(t)}, \tau(R_i, \mathbf{x})\right). \tag{12}$$

The reason we look at $\mathrm{ClipNum}\Big(I_i^{(t)}, \tau(R_i, \mathbf{x})\Big)$ instead of the total number of clipped tuples is that the former is entirely decided locally by $I_i^{(t)}$. In particular, it has low sensitivity:

Lemma 5.4. *Fixed any $\{\tau(R_i, \mathbf{x})\}$ and consider any $\mathbf{I} \sim \mathbf{I}'$ where the only difference happens in $R_{i'}$. For any $t \in \mathbb{Z}^+$, any $i \neq i'$, and any $\mathbf{x} \in \mathcal{B}_{Q,i}$ we have*

$$\mathrm{ClipNum}\Big(I_i^{(t)}, \tau(R_i, \mathbf{x})\Big) = \mathrm{ClipNum}\Big(I_i'^{(t)}, \tau(R_i, \mathbf{x})\Big).$$

For any $\mathbf{x} \subseteq \mathcal{B}_{Q,i'}$, we have

$$\left|\mathrm{ClipNum}\Big(I_{i'}^{(t)}, \tau(R_{i'}, \mathbf{x})\Big) - \mathrm{ClipNum}\Big(I_{i'}'^{(t)}, \tau(R_{i'}, \mathbf{x})\Big)\right| \leq 1.$$

Given the preceding discussions, for each $(i, \mathbf{x}) \in \mathcal{B}_Q$, we can employ SVT to pinpoint the time to double $\tau(R_i, \mathbf{x})$, using the following sensitivity-1 queries

$$f_t(\mathbf{I}) = \mathrm{ClipNum}\big(I_i^{(t)}, \tau(R_i, \mathbf{x})\big) - \left(\frac{8}{\varepsilon}\log(2/\beta) + \frac{8}{\varepsilon}\log(t+1)\right)$$

for successive t with the stopping threshold $\eta = 0$. The negative term in $f_t(\mathbf{I})$ is from Lemma 3.8, which ensures that when the SVT stops, we have $\mathrm{ClipNum}\big(I_i^{(t)}, \tau(R_i, \mathbf{x})\big) > 0$ with probability $1 - \beta$. This means that the current $\mathrm{mf}(I_i^{(t)}, \mathbf{x})$ has exceeded $\tau(R_i, \mathbf{x})$ so it is time to double $\tau(R_i, \mathbf{x})$.

As we will invoke both ClipDP and SVT multiple times, the privacy budget must be allocated properly. First, we equally split the total privacy budget between them. For ClipDP, it is restarted after each doubling of some $\tau(R_i, \mathbf{x})$. For an unbounded stream, this may happen for an unbounded number of times, so we allocate the privacy budget using a telescoping series: $\varepsilon\theta/\big(2(k+1)^{1+\theta}\big)$ is assigned for the kth ClipDP. This ensures that all invocations of ClipDP satisfies $\varepsilon/2$-DP. For SVT, by Lemma 5.4, the difference between two neighboring streams can only affect one relation, so each relation will get a privacy budget of $\varepsilon/2$. Each relation has at most $m_{\max}$ constraints to monitor using SVT, and there can be an unlimited number of SVTs for each constraint, so we assign a privacy budget of $\varepsilon\theta/\big(2m_{\max}(k+1)^{1+\theta}\big)$ to its kth SVT. The detailed algorithm is shown in Algorithm 4, which we denote by DynamicClipDP.

The privacy and utility guarantees of DynamicClipDP are analyzed in the following theorem.

Theorem 5.5. *For any $\varepsilon > 0$,* DynamicClipDP *preserves ε-DP. For any $\beta > 0$, $\theta > 0$, any $\mathbf{I}$, and any $t \in \mathbb{Z}^+$, with probability at least $1 - \beta$, it returns a $\widetilde{Q}(\mathbf{I}^{(t)})$ such that*

$$\left|\widetilde{Q}(\mathbf{I}^{(t)}) - Q(\mathbf{I}^{(t)})\right| \leq O\left(\frac{1}{\varepsilon\theta} \cdot \mathrm{GS}_Q\Big(\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\}\Big) \cdot \log^{1.5+\theta} t \right.$$

$$\left. \sum_{(i,\mathbf{x}) \in \mathcal{B}_Q} \left(\log^{1+\theta}\Big(\mathrm{mf}(I_i^{(t)}, \mathbf{x})\Big) \cdot \log\big(\log(\mathrm{mf}(I_i^{(t)}, \mathbf{x}))/\beta\big)\right)\right).$$

Proof. The privacy guarantee follows from the preceding discussion. Below we analyze the utility.

First, by Lemma 3.8, for any $(i, \mathbf{x}) \in \mathcal{B}_Q$ and any $k \in \mathbb{Z}^+$, kth $\mathrm{SVT}_{(i,\mathbf{x})}$ stops at t' such that with probability $1 - \beta/\big(2m \cdot (k+1)^2\big)$,

$$f_{t'}(G) > -\frac{m_{\max}(k+1)^{1+\theta}}{\varepsilon\theta}\Big(16\log\big(4m(k+1)^2/\beta\big) + 12\log(t+1)\Big),$$

$$f_{t'-1}(G) < \frac{m_{\max}(k+1)^{1+\theta}}{\varepsilon\theta}\Big(16\log\big(4m(k+1)^2/\beta\big) + 12\log(t+1)\Big),$$

which imply

$$\mathrm{ClipNum}\Big(I_i^{(t')}, 2^k\Big) > 0, \tag{13}$$

$$\mathrm{ClipNum}\Big(I_i^{(t'-1)}, 2^k\Big) = O\bigg(\frac{k^{1+\theta}}{\varepsilon\theta}\Big(\log\big(mk/\beta\big) + \log(t)\Big)\bigg) \tag{14}$$

Combining the probabilities across all values of k and all $(i, \mathbf{x}) \in \mathcal{B}_Q$, we have, with probability at least $1 - \beta/2$, (13) and (14) hold for all instances of SVT.

(13) means that for any time $t \in \mathbf{Z}^+$, and any $(i, \mathbf{x}) \in \mathcal{B}_Q$, the $\tau(R_i, \mathbf{x})$ used in BM at time t, must have

$$\tau(R_i, \mathbf{x}) \le 2 \cdot \mathrm{mf}\big(I_i^{(t)}, \mathbf{x}\big), \tag{15}$$

which implies, it corresponds to kth instance of $\mathrm{SVT}_{(i,\mathbf{x})}$ such that

$$k \le \log\big(\mathrm{mf}(I_i^{(t)}, \mathbf{x})\big) + 1. \tag{16}$$

Now, let us analyze the error. For the bias, incorporating (12), (14), and (16), we have, for each time $t \in \mathbb{Z}^+$,

$$\mathrm{ClipNum}\Big(\mathbf{I}^{(t)}, \big\{\tau(R_i, \mathbf{x})\big\}\Big) = O\bigg(\frac{1}{\varepsilon\theta} \sum_{(i,\mathbf{x})\in\mathcal{B}_Q} \bigg(\log^{1+\theta}\Big(\mathrm{mf}(I_i^{(t)}, \mathbf{x})\Big)\Big(\log\big(\log(\mathrm{mf}(I_i^{(t)}, \mathbf{x}))/\beta\big) + \log t\Big)\bigg)\bigg)$$

Further integrating this with the fact that each tuple in $\mathbf{I}^{(t)}$ contributes at most $\mathrm{GS}_Q\Big(\big\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\big\}\Big)$ to $Q(\mathbf{I}^{(t)})$, we get the desired bias.

For the noise component, we draw from equation (16) that for any $t \in \mathbb{Z}^+$, the kth ClipDP is applied under the condition

$$k \le \sum_{(i,\mathbf{x})\in\mathcal{B}_Q} \log\big(\mathrm{mf}(I_i^{(t)}, \mathbf{x})\big) + m. \tag{17}$$

By combining (15) and (17), we have the noise bounded by

$$O\bigg(\frac{1}{\varepsilon\theta} \cdot \mathrm{GS}_Q\Big(\big\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\big\}\Big) \cdot \log^{1.5+\theta} t \sum_{(i,\mathbf{x})\in\mathcal{B}_Q} \log^{1+\theta}\Big(\mathrm{mf}(I_i^{(t)}, \mathbf{x})\Big) \cdot \log(1/\beta)\bigg).$$

□

Optimality. The algorithm DynamicClipDP attains an error of $\tilde{O}\Big(\mathrm{GS}_Q\Big(\big\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\big\}\Big)\Big)$ for each time $t \in \mathbb{Z}^+$ even over an unbounded stream. This matches (up to polylogarithmic factors) the best-known result utilizing the maximum frequency information to calibrate the noise [32, 36] in the static setting, which is equivalent to a finite stream of length t and the query result is released only once at the end.

In the static setting, an $\Omega\Big(\mathrm{GS}_Q\Big(\big\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\big\}\Big)\Big)$ lower bound has been established for certain queries [16, 19, 20]. For example, for the line-2 query $|R_1(A, B) \bowtie R_2(B, C)|$, it has been shown that no DP mechanism can achieve an error lower than

$$\Omega\Big(\max\big(\mathrm{mf}(I_1, B), \mathrm{mf}(I_2, B)\big)\Big),$$

and for the n-star query $|R_1(A, B_1) \bowtie R_2(A, B_2) \bowtie \cdots \bowtie R_n(A, B_n)|$, there is a lower bound of

$$\Omega\left(\prod_i \mathrm{mf}(I_i, A) \Big/ \min_i \mathrm{mf}(I_i, A)\right).$$

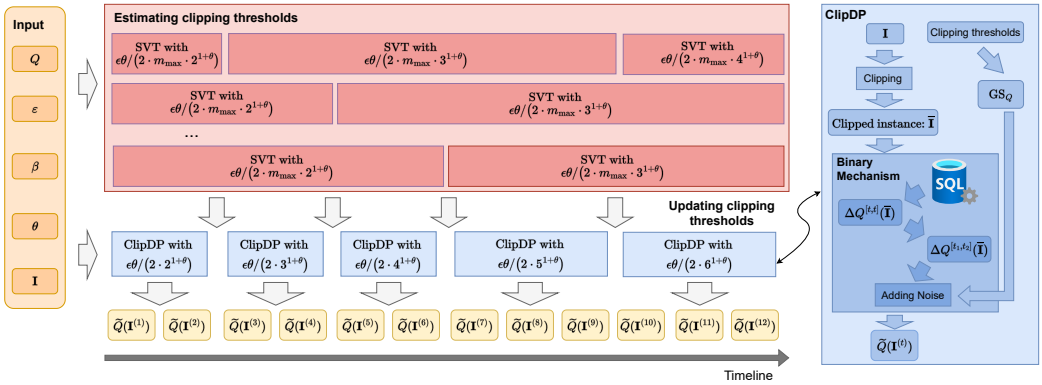


Fig. 5. Implementation of our algorithm

Note that $\mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right)$ is exactly equal to these two expressions in these two cases. However, [19, 20] show for other queries like n-line path counting queries with $n \geq 3$, a better error can be achieved in the static setting. How to achieve those errors in the dynamic setting is still an open problem.

6 EXTENSIONS

Self-joins. For a query with self-joins, we can treat the multiple occurrences of a relation in the query as copies of the same relation and then invoke our algorithm. Accordingly, every incoming tuple will be treated as an insertion to each of the copies. Note that this has an impact on privacy, since two neighboring instances now have a distance of ℓ, where ℓ is the maximum number of occurrences of any relation in the query. Therefore, we need to run our mechanism with a privacy budget of ε/ℓ. The error will then also grow by a factor of ℓ accordingly.

Example 6.1. Consider counting the number of triangles in a directed graph, which can be written as a self-join:

$$Q_\triangledown = \left|\mathrm{Edge}(A,B) \bowtie \mathrm{Edge}(B,C) \bowtie \mathrm{Edge}(C,A)\right|. \tag{18}$$

We first rewrite the query into the standard triangle query in (1) by instituting three distinct relations R_1, R_2, and R_3, all of which are copies of Edge. For each incoming tuple e, we will insert e into each of R_1, R_2, R_3. We have $\ell = 3$ for this query. □

Predicates. As shown in prior work [32, 36], the presence of predicates does not increase the sensitivity of the query. Therefore, we can compute GS_Q of the query as before, while ignoring the predicates. In addition, it is clear that the predicates do not affect the monotonicity and supermodularity of the query. The only change is that, in line 3 of BM (Algorithm 2) when we evaluate the true delta query $\Delta Q^{[t_1,t_2]}(\mathbf{I})$, we need to apply the predicates.

7 IMPLEMENTATION

Our algorithm is versatile and can be implemented on top of any SQL query engine, with architecture illustrated in Figure 5. The algorithm consists of two main components. The first component is to dynamically estimate the clipping thresholds $\left\{\tau(R_i, \mathbf{x})\right\}$. During this phase, $|\mathcal{B}_Q|$ instances of SVT are executed concurrently where each instance is designated to detect the time step to double the value for a unique $(i, \mathbf{x}) \in \mathcal{B}_Q$. Subsequent to each doubling, the SVT is re-initialized with a diminished privacy budget.

The second part of the algorithm is designed to execute ClipDP utilizing the dynamically estimated clipping thresholds $\{\tau(R_i, \mathbf{x})\}$ to answer the query at each time step. In this phase, we initially clip the instance $\mathbf{I}$ to $\bar{\mathbf{I}}$ and compute GS_Q using these thresholds $\{\tau(R_i, \mathbf{x})\}$. Subsequently, both $\bar{\mathbf{I}}$ and GS_Q are inputted into the binary mechanism. Within the binary mechanism, at each time t, $\Delta Q^{[t,t]}(\bar{\mathbf{I}})$ is formulated as a multi-way join counting query. Assuming the tuple $e^{(t)}$ is from the relation R_i, then,

$$\Delta Q^{[t,t]}(\bar{\mathbf{I}}) = \left| e^{(t)} \bowtie \left(\bowtie_{j \neq i} I_j^{(t-1)} \right) \right|.$$

In our implementation, we frame the above query as a SQL query. It is worth mentioning that several techniques, as cited [7, 11, 54], can evaluate the above query in more efficient way. Following this, we can conveniently obtain $\Delta Q^{[t_1,t_2]}(\bar{\mathbf{I}})$ using these instances of $\Delta Q^{[t,t]}(\bar{\mathbf{I}})$. Furthermore, as argued in [10], we only need to maintain at most $\log(t)$ number of $\Delta Q^{[t_1,t_2]}(\bar{\mathbf{I}})$ at each time t. Importantly, every time there is an update in $\{\tau(R_i, \mathbf{x})\}$, a restart is required for this process.

Optimization. For conceptual simplicity, after doubling a $\tau(R_i, \mathbf{x})$ at time t, we restart both the SVT associated with $\tau(R_i, \mathbf{x})$ and the ClipDP mechanism from the beginning of the stream. However, in the actual implementation, this can be avoided by utilizing the information gathered in preceding steps. First, the new instance of SVT needs to track the number of tuples that ought to be clipped at the current time, with the updated clipping threshold. Instead of rewinding the entire instance, a more efficient strategy is to ascertain whether the tuples, previously clipped with the old threshold, will be maintained under the new threshold. Second, during the re-instantiation of the binary tree, we can compress the entire time frame $[1, t]$ into a single time step, given that there is no requirement to answer queries prior to time t.

Computational Complexity and Space Usage. By maintaining all $\mathrm{freq}(I_i^{(t)}, a) > 0$ for $(i, \mathbf{x}) \in \mathcal{B}$ and $a \in \mathbf{dom}(\mathbf{x})$, which requires at most linear space cost, we only require constant running time to execute SVT's and do the clipping at each time step. Furthermore, as corroborated in the proof of Theorem 5.5, for any time $t \in \mathbb{Z}^+$, we only need to store at most $\tilde{O}(1)$ clipped tuples, implying a logarithmic running time for restarting an SVT. Given that there are at most $O(\log(t))$ restarts of SVT before reaching time t, the amortized cost of these restarts becomes $o(1)$. In the binary mechanism, aside from computing $\Delta Q^{[t,t]}(\bar{\mathbf{I}})$, only a constant running time and $\log(t)$ memory are used at any given time t. Every re-building process requires the computation of the query update, influenced by some previously clipped tuples. Since each tuple is inserted into the database only once, the amortized cost for any time t aligns with $\Delta Q^{[t,t]}(I)$, equivalent to the non-private setting. Above all, in addition to computing $\Delta Q^{[t,t]}(\bar{\mathbf{I}})$, which is needed even for non-private continual observation, our system incurs a constant amortized computational overhead at each time step with linear space usage.

8 EXPERIMENTS

In this section, we compared our mechanism in Section 5 with the following three baselines on both graph pattern counting queries and general multi-way join counting queries in the dynamic setting.

Composition: We employed advanced composition [26] to allocate the privacy budget ε', where $\varepsilon = \sqrt{2T \ln(1/\delta)}\varepsilon' + T\varepsilon'(e^{\varepsilon'} - 1)$ to each $\mathbf{I}^{(t)}$, on which we use the *residual sensitivity* (RS) mechanism [19, 20], the state-of-the-art algorithm for static multi-way join counting queries under DP.

Dataset	Simulated-temporal networks.			Real-temporal networks.		
	RoadnetUS	**DuWiki**	**CaWiki**	**Dblp**	**Flickr**	**StackOverflow**
Number of edges	4.08×10^7	1.08×10^7	1.14×10^7	9.37×10^6	4.27×10^6	9.22×10^6
Maximum degree	16	167	442	122	76	158

Table 4. Network datasets used in the experiments.

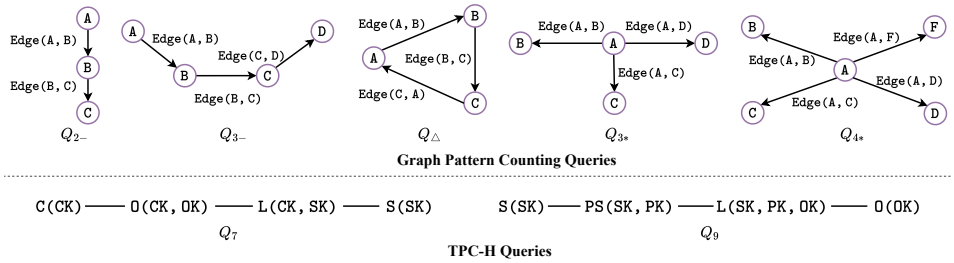


Fig. 6. The query structures.

Binary mechanism. This is the mechanism described in Section 4, which requires the frequency constraints $\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\}$. In our experiments, all datasets have a temporal domain ranging between 2 million and 50 million with a maximum frequency of up to 2^{10}, so we set each $\widehat{\mathrm{mf}}(R_i, \mathbf{x})$ to 2^{15}.

Clipping mechanism. This is the clipping mechanism discussed in Section 5.1 but with fixed clipping thresholds $\{\tau(R_i, \mathbf{x})\}$. As there is no method prior to this work on how to determine the clipping threshold, we randomly selected a value from $\{2, 4, 8, \ldots, 2^{15}\}$ and set all $\tau(R_i, \mathbf{x})$ to that value.

8.1 Setup

Query. For graph pattern counting queries, we used five queries: length-2 path counting query Q_{2-}, length-3 path counting query Q_{3-}, triangle counting query $Q_{\triangledown}$, 3-star counting query Q_{3*}, and 4-star counting query Q_{4*}. Note that for all graph pattern counting queries, we use predicates to avoid duplicate-counting. For example, we equip $Q_{\triangledown}$ defined in (18) with A < B and A < B. For multi-way join counting queries, we used two queries from the TPC-H benchmark, Q_7 and Q_9, omitting the projection and group-by clauses. In addition, to avoid exceedingly small join results, we excluded the predicates for Q_9. For Q_7, we maintain the predicate on shipdate within a four-year time range. Furthermore, we also utilized the primary key information in the TPC-H schemas by setting $\widehat{\mathrm{mf}}(\mathsf{C}, \mathsf{CK})$, $\widehat{\mathrm{mf}}(\mathsf{S}, \mathsf{SK})$, and $\widehat{\mathrm{mf}}(\mathsf{O}, \mathsf{OK})$ to be 1.

Dataset. For graph pattern counting queries, we used 6 real world networks datasets categorized into two classes based on the presence of timestamps on each edge, referred to as *real-temporal networks* and *simulated-temporal networks*. For simulated-temporal networks, edges were allocated a random order. We used three datasets. **RoadnetUS** corresponds to the road network of the USA and was used in the 9th DIMACS Implementation Challenge [37]. **CaWiki** and **DuWiki** characterize the hyperlink networks between Wikipedia articles written in the Catalan and Dutch languages respectively [37]. For real-temporal network, we selected three datasets, retaining their intrinsic temporal sequence to organize the edges. **Dblp** stands for the collaboration graph of authors contributing to the DBLP computer science bibliography [40]. **Flickr** is the social network of Flickr users [43]. **StackOverflow** is the interaction network of users from the Stack Exchange

Dataset		Simulated-temporal data						Real-temporal data					
		RoadnetUS		**DuWiki**		**CaWiki**		**Dblp**		**Flickr**		**StackOverflow**	
		RE(%)	RT(s)	RE(%)	RT(s)	RE(%)	RT(s)	RE(%)	RT(s)	RE(%)	RT(s)	RE(%)	RT(s)
Q_{2-}	Ours	0.0065	6.79	0.172	2.55	0.254	2.98	0.119	3.11	0.34	1.17	0.19	1.91
	BM	3.08	8.54	2.09	3.84	0.882	2.42	3.13	3.22	15.4	1.41	15.1	2.78
	CM	0.107	7.03	3.54	3.47	14.8	3.49	1.95	3.34	2.21	1.38	2.89	3.17
	RS	4.25×10^4	42.2	9.32×10^3	14.4	4.84×10^3	12.3	1.19×10^4	13.3	3.79×10^4	9.38	1.2×10^4	13.5
Q_{3-}	Ours	0.145	17.3	3.7	6.82	7.6	5.55	3.61	11.2	29.7	6.6	3.74	12.2
	BM	6.06×10^4	14.8	6.03×10^3	10.9	1.06×10^3	5.39	9.95×10^3	11.7	1.6×10^5	6.8	1.03×10^4	10.8
	CM	212	18.7	54.5	10.7	52.4	5.74	69.5	12.3	1.37×10^3	6.91	63.2	10.9
	RS	9.42×10^9	160	1.49×10^8	109	2.73×10^7	186	1.63×10^8	82.4	1.81×10^9	38.2	1.75×10^8	80.2
$Q_{\triangledown}$	Ours	1.37	12.4	4.41	6.92	7.63	11.4	1.14	9.01	30.1	6.78	0.318	12.2
	BM	272	13.1	12.2	10.3	4.6	9.51	5.04	12.5	375	6.35	1.21	11
	CM	10.7	14.3	11.3	13.7	23.9	7.46	3.71	13.3	32	7.17	5.18	12.2
	RS	1.37×10^{12}	114	6.37×10^9	113	3.19×10^9	200	3.14×10^9	78.4	9.03×10^{10}	42.2	8.57×10^8	76
Q_{3*}	Ours	0.00302	8.23	0.377	2.07	0.554	1.15	0.263	1.69	2.16	1.75	0.337	1.9
	BM	2.61×10^3	9.19	1.21×10^4	1.54	210	1.31	1.45×10^3	1.82	5.43×10^4	2.38	1.38×10^3	2.22
	CM	11.6	8.21	33.1	1.51	34.3	1.22	33	1.95	149	2.01	34.8	1.94
	RS	3.63×10^8	72.3	2.34×10^7	34.3	4.85×10^6	33	3.4×10^7	30.9	3.64×10^8	20.6	3.34×10^7	30.3
Q_{4*}	Ours	0.0262	12	1.89	3.02	2.83	2.1	1.15	2.45	12.6	2.57	1.36	3
	BM	4.12×10^7	12.3	1.4×10^6	2.25	1.32×10^5	2.05	3.55×10^6	2.74	2.11×10^8	2.36	2.16×10^6	2.93
	CM	2.46×10^4	9.77	1.07×10^3	2.28	104	1.62	2.56×10^3	2.58	9.43×10^4	2.79	2.31×10^3	2.82
	RS	1.03×10^{14}	253	3.56×10^{11}	133	2.03×10^{10}	133	7.31×10^{11}	122	9.25×10^{12}	83.1	3.92×10^{11}	119

Table 5. Comparison among our mechanism, residual sensitivity with the advanced composition (RS), binary mechanism (BM), and clipping mechanism (CM) on graph pattern counting queries ($\varepsilon = 4$). RE and RT denote relative error and running time and we report median of RE over all selected timestamps.

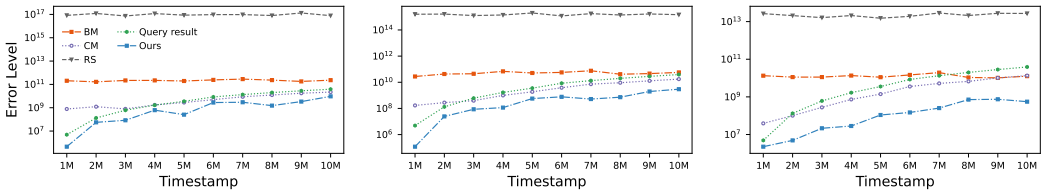
website, Stack Overflow [38]. All the aforementioned networks are undirected. Among them, **StackOverflow** was collected from SNAP [39] while the remaining were from KONECT [37]. To avoid too much error, we have deleted the top 5% nodes with the highest degrees. Detailed information regarding these datasets can be found in Table 4. For TPC-H data, we used the dataset of scale 10, which encompasses about 75 million tuples. Given the absence of timestamps on these tuples, we also arranged them with a random order with tuples in come before.

Experimental parameters. All experiments were conducted on a Linux server equipped with a 24-core 48-thread 2.2GHz Intel Xeon CPU and 256GB of memory. We used the the absolute difference between the actual result and the DP result as the error metric. Each experiment was repeated 20 times,[2] with error being recorded every 5×10^5 time steps. For each selected time step, we excluded the 20% largest and 20% smallest errors and reported the average error for the remaining runs. We set the privacy budget at $\varepsilon = 1, 4, 16$, with the default value being 4. RS has an input as δ, which is set to 10^{-10}. Additionally, the failure probability $\beta = 0.1$ and $\theta = 1$. Furthermore, since RS requires knowledge of the length of the time domain in advance, for the sake of fairness, we employed the finite domain version of the binary mechanism. This version was also used to construct both our mechanism and the clipping mechanism.

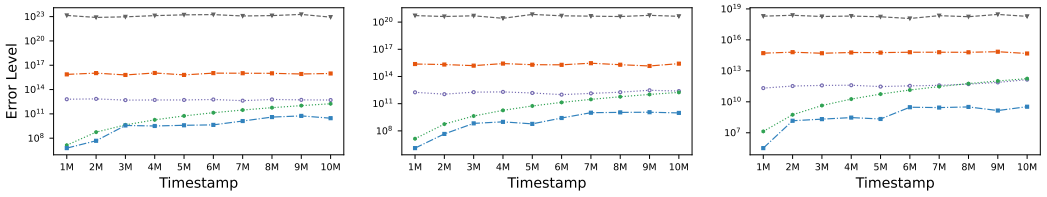
8.2 Graph Pattern Counting Queries

Utility and efficiency. The errors and running times of all mechanisms pertaining to graph pattern counting queries are shown in Table 5. For each selected timestamp, we collected the relative error and report their median. The results indicate a clear superiority of our mechanism in terms of the utility: our mechanism consistently exhibits high utility, with a median relative error of under 10% in all experiments except three queries over **Flickr** dataset, where the median of relative error is at most 30%. That is because **Flickr** dataset is sparser thus has a smaller graph pattern count, amplifying the relative error. In a comparative perspective, RS loses utility across all experiments,

[2]For the clipping mechanism, the number of repetitions was increased to 100 since it has more randomness.



(a) Q_{3-} on **CaWiki**.



(b) Q_{4*} on **DuWiki**.

Fig. 7. Error levels vs time of various mechanisms with $\varepsilon = 1, 4, 16$
. The actual query result is also piloted to help see whether the mechanisms have utility.

aligning with our analysis that its error bears a polynomial dependency on the time domain size T. BM and CM always have much higher error level than our mechanism: our improvement in error over BM and CM can be as large as 10^6 and 10^{10} respectively. One exception happens for the Q_{∇} over **CaWiki** dataset, where the ratio between $\mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right)$ and $\mathrm{GS}_Q\left(\left\{\widehat{\mathrm{mf}}(R_i, \mathbf{x})\right\}\right)$ is only a large constant in most time steps. Importantly, as query complexity increase, their performance gap from our mechanism also increases, confirming our theoretical analysis. Additionally, in most cases, especially when queries are complex or datasets are sparse, CM outperforms BM due to it removes the error dependency on prior frequency constraints and can utilizes a smaller clipping thresholds to reduce the noise. However, for simple query like Q_{2-} over dense datasets, the clipping could lead to a significant bias. That also demonstrate the importance to estimate the actual maximum frequency adaptvely to balance the bias and noise. In terms of the running time, BM, CM, and our mechanism have similar running times, all of which are much smaller than RS, matching our analysis that BM, CM, and our mechanism have a similar running time as non-private mechanism.

Error with Time. We also conducted experiments to evaluate how the error changes with the time for various mechanisms with different $\varepsilon = 1, 4, 16$. Here, we plot the results for Q_{3-} on the **CaWiki** dataset and Q_{4*} on the **DuWiki** dataset in Figure 7. The results show that our mechanism, consistently has a high utility except on the initial time steps, where the query result is very small. Moreover, benefiting from the adaptive estimation of the actual maximum frequency, our mechanism has the time-specific error. Contrarily, the errors of the other three mechanisms do not have a strong correlation with the time. It matches our theoretical guarantee that we attain the error proportional to $\mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right)$ at each time while their errors depend on T or some predefined frequency constraints.

Comparison with binary mechanism under different frequency constraints. In the next set of experiments, we compare our mechanism against the binary mechanism under different predefined frequency constraints. We tested the query Q_{3*} on dataset **StackOverflow** with a frequency constraint ranging from 2^{10} to 2^{15}. As mentioned in Section 8.1, 2^{10} closely approximates the actual maximum frequency of the dataset. The results are shown in Figure 8. The findings firstly demonstrate that the

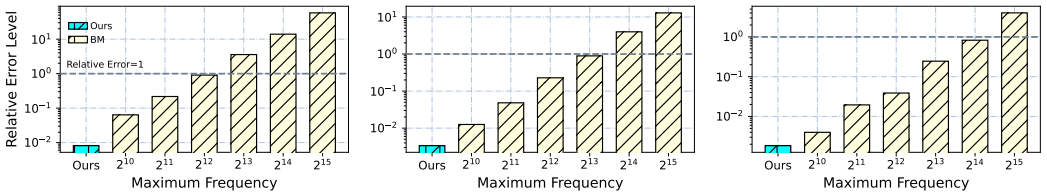


Fig. 8. Error levels of our mechanism and binary mechanism (BM) under different maximum frequency ranging from 2^{10} to 2^{15} on Q_{3*} over **StackOverflow** with $\varepsilon = 1, 4, 16$.

.

Mechanism		Ours		BM		CM		RS	
		RE(%)	RT(s)	RE(%)	RT(s)	RE(%)	RT(s)	RE(%)	RT(s)
Result	Q_7	21.2	3.19	1.18×10^5	2.92	192	2.27	1.68×10^{17}	392
	Q_9	18.1	1.32	1.43×10^5	3.19	142	1.49	1.98×10^{17}	848

Table 6. Comparison among our mechanism, residual sensitivity with the advanced composition (RS), binary mechanism (BM), and clipping mechanism (CM) on TPC-H queries ($\varepsilon = 4$). RE and RT denote relative error and running time.

error of binary mechanism highly depends on the setting of frequency constraints. Moreover, our mechanism consistently outperforms the binary mechanism, even when the frequency constraint is nearly equivalent to the actual maximum frequency. That is because our mechanism achieves an time-specific error, i.e., an error proportional to $\mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(t)}, \mathbf{x})\right\}\right)$ at at each time t. In contrast, the binary mechanism maintains an error proportional to $\mathrm{GS}_Q\left(\left\{\mathrm{mf}(I_i^{(T)}, \mathbf{x})\right\}\right)$ across all time steps. Although both mechanisms eventually reach a similar error level, our method performs better at intermediate time steps. Given that we report the average error over all time steps, our mechanism attains a significantly lower error. This outcome also reveals the advantage of adaptively estimating maximum frequencies.

8.3 Multi-way Join Counting Queries

We also evaluated all mechanisms using two selected TPC-H queries, with the results presented in Table 6. Similar to the graph pattern counting queries, our mechanism consistently outperforms the others. However, compared with graph pattern counting queries, our improvement over BM and CM is more modest. This is because as mentioned in Section 8.1, the primary key information in the TPC-H schema will be used. Primary key constraints can be regarded as some strong prior knowledge of frequency constraints thus BM and CM can benefit a lot. On the other hand, our mechanism will not be affected since it automatically adapts to the actual maximum frequencies. Above all, the errors of BM and CM will be reduced after considering primary key constraints but our errors remain unchanged. As a result, our improvement over BM and CM decreases.

ACKNOWLEDGMENTS

This work has been supported by HKRGC under grant numbers 16205420, 16205422, and 16204223, National Science Foundation under grant numbers 2128519 and 2044679, a grant from ONR, a grant from the DARPA SIEVE program under a subcontract from SRI, a gift from Cisco, and a Packard Fellowship. We would also like to thank the anonymous reviewers who have made valuable suggestions on improving the presentation of the paper.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.

[2] Mahmoud Abo Khamis, Hung Q Ngo, and Dan Suciu. 2017. What do Shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another?. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 429–444.

[3] Myrto Arapinis, Diego Figueira, and Marco Gaboardi. 2016. Sensitivity of Counting Queries. In *International Colloquium on Automata, Languages, and Programming (ICALP)*.

[4] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. 2013. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. 87–96.

[5] Jean Bolot, Nadia Fawaz, Shanmugavelayutham Muthukrishnan, Aleksandar Nikolov, and Nina Taft. 2013. Private decayed predicate sums on streams. In *Proceedings of the 16th International Conference on Database Theory*. 284–295.

[6] Kuntai Cai, Xiaokui Xiao, and Graham Cormode. 2023. Privlava: synthesizing relational data with foreign keys under differential privacy. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.

[7] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering* 38, 4 (2015).

[8] Adrian Rivera Cardoso and Ryan Rogers. 2022. Differentially private histograms under continual observation: Streaming selection into the unknown. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2397–2419.

[9] T-H Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. 2012. Differentially private continual monitoring of heavy hitters from distributed streams. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 140–159.

[10] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and Continual Release of Statistics. *ACM Transactions on Information and System Security* (2011).

[11] Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Robert DeLine, Danyel Fisher, John C Platt, James F Terwilliger, and John Wernsing. 2014. Trill: A high-performance incremental query processor for diverse analytics. *Proceedings of the VLDB Endowment* 8, 4 (2014), 401–412.

[12] Shixi Chen and Shuigeng Zhou. 2013. Recursive mechanism: towards node differential privacy and unrestricted joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 653–664.

[13] Yan Chen, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2017. Pegasus: Data-adaptive differentially private stream processing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1375–1388.

[14] Rachel Cummings, Sara Krehbiel, Kevin A Lai, and Uthaipon Tantipongpipat. 2018. Differential privacy for growing databases. *Advances in Neural Information Processing Systems* 31 (2018).

[15] Sergey Denisov, Brendan McMahan, Keith Rush, Adam Smith, and Abhradeep Thakurta. 2022. Improved differential privacy for sgd via optimal private linear operators on adaptive streams. In *NeurIPS*.

[16] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[17] Wei Dong, Qiyao Luo, and Ke Yi. 2023. Continual Observation under User-level Differential Privacy. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2190–2207.

[18] Wei Dong, Dajun Sun, and Ke Yi. 2023. Better than Composition: How to Answer Multiple Relational Queries under Differential Privacy. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.

[19] Wei Dong and Ke Yi. 2021. Residual Sensitivity for Differentially Private Multi-Way Joins. In *Proc. ACM SIGMOD International Conference on Management of Data*.

[20] Wei Dong and Ke Yi. 2022. A Nearly Instance-optimal Differentially Private Mechanism for Conjunctive Queries. In *Proc. ACM Symposium on Principles of Database Systems*.

[21] Wei Dong and Ke Yi. 2023. Query Evaluation under Differential Privacy. *ACM SIGMOD Record* 52, 3 (2023), 6–17.

[22] Wei Dong and Ke Yi. 2023. Universal private estimators. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 195–206.

[23] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 715–724.

[24] Cynthia Dwork, Moni Naor, Omer Reingold, and Guy N Rothblum. 2015. Pure differential privacy for rectangle queries via private partitions. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 735–751.

[25] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N Rothblum, and Salil Vadhan. 2009. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the forty-first annual ACM symposium*

on Theory of computing. 381–390.
[26] Cynthia Dwork and Aaron Roth. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
[27] Juanru Fang, Wei Dong, and Ke Yi. 2022. Shifted Inverse: A General Mechanism for Monotonic Functions under User Differential Privacy. (2022).
[28] Hendrik Fichtenberger, Monika Henzinger, and Wolfgang Ost. 2021. Differentially Private Algorithms for Graphs Under Continual Observation. In *29th Annual European Symposium on Algorithms (ESA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
[29] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. 2012. Size and treewidth bounds for conjunctive queries. *Journal of the ACM (JACM)* 59, 3 (2012), 1–35.
[30] Monika Henzinger and Jalaj Upadhyay. 2022. Constant matters: Fine-grained Complexity of Differentially Private Continual Observation Using Completely Bounded Norms. *arXiv preprint arXiv:2202.11205* (2022).
[31] Monika Henzinger, Jalaj Upadhyay, and Sarvagya Upadhyay. 2022. Almost tight error bounds on differentially private continual counting. *arXiv preprint arXiv:2211.05006* (2022).
[32] Noah Johnson, Joseph P Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proceedings of the VLDB Endowment* 11, 5 (2018), 526–539.
[33] Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. 2021. Practical and private (deep) learning without sampling or shuffling. In *International Conference on Machine Learning*. PMLR, 5213–5225.
[34] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. 2011. Private analysis of graph structure. *Proceedings of the VLDB Endowment* 4, 11 (2011), 1146–1157.
[35] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2013. Analyzing graphs with node differential privacy. In *Theory of Cryptography Conference*. Springer, 457–476.
[36] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: a differentially private SQL query engine. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1371–1384.
[37] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*. 1343–1350.
[38] Jure Leskovec and Andrej Krevl. 2014. SNAP: Stanford network analysis project.
[39] Jure Leskovec and Andrej Krevl. 2016. SNAP datasets: Stanford large network dataset collection (2014). *URL http://snap. stanford. edu/data* (2016), 49.
[40] Michael Ley. 2002. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *International symposium on string processing and information retrieval*. Springer, 1–10.
[41] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. 2015. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB journal* 24, 6 (2015), 757–781.
[42] Frank D McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 19–30.
[43] Alan Mislove, Hema Swetha Koppula, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2008. Growth of the flickr social network. In *Proceedings of the first workshop on Online social networks*. 25–30.
[44] Arjun Narayan and Andreas Haeberlen. 2012. DJoin: Differentially private join queries over distributed databases. In *USENIX Symposium on Operating Systems Design and Implementation*. 149–162.
[45] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2007. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. 75–84.
[46] Catuscia Palamidessi and Marco Stronati. 2012. Differential Privacy for Relational Algebra: Improving the Sensitivity Bounds via Constraint Systems. In *QAPL*.
[47] Victor Perrier, Hassan Jameel Asghar, and Dali Kaafar. 2019. Private continual release of real-valued data streams. In *26th Annual Network and Distributed System Security Symposium, NDSS 2016*. Internet Society, 1–13.
[48] Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2014. Calibrating Data to Sensitivity in Private Data Analysis. *Proceedings of the VLDB Endowment* 7, 8 (2014).
[49] Yuan Qiu and Ke Yi. 2022. Differential Privacy on Dynamic Data. *arXiv preprint arXiv:2209.01387* (2022).
[50] Shuang Song, Susan Little, Sanjay Mehta, Staal Vinterbo, and Kamalika Chaudhuri. 2018. Differentially private continual release of graph statistics. *arXiv preprint arXiv:1809.02575* (2018).
[51] Dajun Sun, Wei Dong, and Ke Yi. 2023. Confidence Intervals for Private Query Processing. *Proceedings of the VLDB Endowment* 17, 3 (2023), 373–385.
[52] Yuchao Tao, Xi He, Ashwin Machanavajjhala, and Sudeepa Roy. 2020. Computing Local Sensitivities of Counting Queries with Joins. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 479–494.

[53] Jalaj Upadhyay. 2019. Sublinear space private algorithms under the sliding window model. In *International Conference on Machine Learning*. PMLR, 6363–6372.
[54] Qichen Wang, Xiao Hu, Binyang Dai, and Ke Yi. 2023. Change Propagation Without Joins. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1046–1058.
[55] Tianhao Wang, Joann Qiongna Chen, Zhikun Zhang, Dong Su, Yueqiang Cheng, Zhou Li, Ninghui Li, and Somesh Jha. 2021. Continuous release of data streams under both centralized and local differential privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1237–1253.
[56] Bing Zhang, Vadym Doroshenko, Peter Kairouz, Thomas Steinke, Abhradeep Thakurta, Ziyin Ma, Himani Apte, and Jodi Spacek. 2023. Differentially Private Stream Processing at Scale. *arXiv preprint arXiv:2303.18086* (2023).
[57] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2015. Private release of graph statistics using ladder functions. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 731–745.

Received October 2023; revised January 2024; accepted February 2024