Better than Composition: How to Answer Multiple Relational Queries under Differential Privacy

Wei Dong  Dajun Sun  Ke Yi
Hong Kong University of Science and Technology
Hong Kong, China
{wdongac,dsunad,yike}@cse.ust.hk

ABSTRACT

Answering relational queries under differential privacy has attracted a lot of attention in recent years due to growing concerns on personal privacy, and instance-optimal mechanisms have been developed for a single query. However, most real-world data analytical tasks require multiple queries to be answered under a total privacy budget. The standard solution to extend the single-query mechanism to multiple queries is via privacy composition. However, we observe that this may yield an error bound that could be a $\sqrt{d}$-factor worse from the optimal, where $d$ is the number of queries. In this paper, we present a different, more holistic approach that closes this gap. In addition to theoretical optimality, our new mechanism also significantly outperforms privacy composition in practice, especially on more skewed data and large $d$.

CCS CONCEPTS

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

KEYWORDS

Differential privacy; SJA query; Multiple queries

ACM Reference Format:

1 INTRODUCTION

Query answering under differential privacy (DP) has been studied extensively in the last 20 years. To date, most problems over a flat table have been relatively well solved. A single (counting or linear) query can be easily answered by the classical Laplace mechanism or the Gaussian mechanism [17], so most efforts have been devoted to the problem of answering a set of $d$ queries. There are two general approaches to this multi-query problem. The first is privacy composition, i.e., we divide the privacy budget to the $d$ queries and answer each query with the single-query mechanism. Using advanced composition [18], the utility suffers an $O(\sqrt{d})$-factor degradation\footnote{The $\tilde{O}$ notation suppresses the dependency on $\epsilon$ and polylogarithmic factors.}, which is the best we can if the queries are arbitrary linear queries, unless $d$ is larger than the size of the database [19]. The other approach is to exploit some special structures of the queries. For instance, if all queries are ranges queries, then the $O(\sqrt{d})$-factor degradation can be reduced to $O(1)$ [15, 29, 41].

1.1 DP in Relational Databases

The situation becomes more complicated in a relational database with multiple relations (tables), mostly due to two challenges.

Challenge 1: Unbounded global sensitivity. First, unlike a flat table, a relational schema models complicated relationships among different types of entities, not all of which are equally positioned in terms of privacy protection. Consider the TPC-H schema. We usually take Customer or Supplier (or both) as the private entities, often called “users”, while Nation and Region are public. Furthermore, any tuple that has a foreign key (FK) reference (directly or indirectly) to a user, such as a lineitem in an order placed by a customer, is considered as data belonging to the user. This results in what is known as user-level DP, as opposed to tuple-level DP used over a flat table. The formal user-level DP definition in relational databases is deferred to Section 3.1; intuitively, this creates a difficulty to the Laplace/Gaussian mechanism, since two neighboring instances that differ by one user (e.g., a customer or a supplier) may differ by arbitrarily many tuples (e.g., lineitems), thus making the global sensitivity of the query, $DS_Q$, unbounded.

When $DS_Q$ is unbounded, a common approach is to consider the local sensitivity [40], or more appropriately, the downward sensitivity [10]. Informally (formal definition given in Section 3.3), for a given query $Q$, the downward sensitivity of a user $u$, denoted $DS_Q(u)$, is the contribution to $Q$ from all the data belonging to $u$. The downward sensitivity of a database instance $I$, denoted $DS_Q(I)$, is the maximum $DS_Q(u)$ over all $u \in I$. Then, one may use the truncation mechanism [27]: For a truncation threshold $r$, delete all users $u$ and their data where $DS_Q(u) > r$, and then apply the Laplace/Gaussian mechanism with sensitivity $r$. The optimal value of $r$ is $DS_Q(I)$, which results in an instance-optimal error of $O(DS_Q(I))$. However, but this optimal truncation threshold cannot be used directly as it is sensitive to $I$. Thus, the main challenge is to find a near-optimal $r$ in a differentially private manner.

Challenge 2: Self-joins. When the query $Q$ has no self-joins, the truncation mechanism would work (after finding a good $r$). However, self-joins create another challenge, as they introduce correlations among the users: Truncating one user’s data may change the
downward sensitivities of other users, this issue has been identified in \cite{10}, who then propose a linear-program-based solution that fixes the problem while achieving the optimal error $\tilde{O}(DS_Q(I))$. Furthermore, as explained in Section 3.2, the issue with self-joins is more prominent under DP: many queries without explicit self-joins actually induce implicit self-joins due to the privacy requirement. In particular, self-joins are always introduced when the a join result references more than one user, which is a common scenario in relational databases.

### 1.2 From a Single Query to Multiple Queries

Answering a single query is not very useful in practice. Thus, as with the flat table case, it is natural to consider the multi-query problem in relational databases, which includes group-by queries as an important special case (i.e., each group corresponds to one query). Let $Q = (Q_1, \ldots, Q_d)$ be the $d$ queries we wish to answer privately. We use the standard metric of root-mean-square error (RMSE) to measure the utility:

$$\|\tilde{Q}(I) - Q(I)\| = \sqrt{\sum_{k=1}^{d} (\tilde{Q}_k(I) - Q_k(I))^2},$$

or equivalently, the $\ell_2$ distance between the privatized query answers $\tilde{Q}(I)$ and the true answers $Q(I)$, both taken as $d$-dimensional vectors. The notation $\| \cdot \|$ refers to the $\ell_2$ norm of a vector throughout the paper.

In this paper, we will allow the queries to contain arbitrary joins and selection predicates. As we do not assume any restricted form of the joins and predicates, the only known solution is to use privacy composition. By advanced composition, we can allocate a privacy budget of $\tilde{O}(1/\sqrt{d})$ to each query and invoke the single-query mechanism in \cite{10}. This leads to an error of $\tilde{O}(\sqrt[2]{DS_{Qk}(I)})$ for $Q_k$, hence an RMSE of

$$\tilde{O}(\sqrt[2]{d} \cdot DS_{Q_k}(I)^2).$$

**Challenge/opportunity 3: Better than composition.** We make the crucial observation that the error bound in (1) is not optimal. In Section 4, we show that the lower bound for the multi-query problem is $\tilde{O}(\sqrt[2]{d} \cdot DS_Q(I))$, where $DS_Q(I)$, informally speaking, is the largest contribution of any user in $I$ to the $d$ query results measured in $\ell_2$ norm. Note that we have the following relationship:

$$DS_Q(I) \leq \sqrt{\sum_{k=1}^{d} DS_{Q_k}(I)^2} \leq \sqrt[2]{d} \cdot DS_Q(I).$$

Both inequalities are tight: The first inequality becomes an equality if the user with the maximum contribution to $Q$ happens to be the maximum-contribution user to every $Q_k \in Q$, and the second inequality becomes an equality if each user contributes to only one query in $Q$. For typical database instances and queries (especially a group-by query), the situation will be more towards the latter, i.e., each user contributes to a small number of queries (groups), in which case the error bound of (1) can be a $\sqrt[2]{d}$-factor away from optimal. This creates a third challenge, or rather, an opportunity for the multi-query problem, i.e., how to do better than privacy composition.

### 1.3 Our Results

Our key insight is that answering all $d$ queries as a whole can yield a much better result. We start by considering multiple self-join-free queries. We observe that $d$ such queries are equivalent to the sum (mean) estimation problem in $d$ dimensions, a problem that has been extensively studied in the machine learning literature \cite{5, 21, 23}. Restated in our terminology, their algorithms achieve the optimal error of $O(\sqrt[2]{d} \cdot DS_Q(I))$, modulo polylogarithmic factors. However, they are all restricted to instances $I$ in which no user has contribution more than $GS_Q$ for some predefined $GS_Q$, and the hidden logarithmic factors depend on $GS_Q$. More precisely, the best error obtained so far \cite{21} is

$$O\left(\frac{\sqrt[2]{d} \cdot DS_Q(I)}{\sqrt[2]{\log(1/\delta)/\epsilon}}\right),$$

where $\epsilon, \delta$ are the privacy parameters (see Section 3.3). Thus, they do not satisfactorily solve challenge 1. Our first result in this paper is the complete removal of the dependency on $GS_Q$, i.e., we do not impose any restrictions on the database instance $I$, effectively allowing $GS_Q = \infty$. Specifically, in Section 4 we design an algorithm that achieves an error of

$$O\left(\frac{\sqrt[2]{d} \cdot DS_Q(I)}{\sqrt[2]{\log(1/\delta) + \log (DS_Q(I))/\epsilon}}\right).$$

Note that even assuming a finite $GS_Q$, the error bound of (3) is better than (2) since $DS_Q(I) < GS_Q$ by definition. The key to obtaining this result is to find a near-optimal truncation threshold $r$ under an unbounded $GS_Q$, and then the standard truncation mechanism can be applied.

Our main technical innovation is how to deal with self-joins. Self-joins are difficult to handle, since they make the truncation mechanism fail. To tackle self-joins, R2T \cite{10} uses a series of linear programs (LPs), which can be considered as the more general forms of the LPs used in private graph analysis \cite{25}. However, as we explain in Section 5.1, these LPs do not work for multiple queries due to fundamental reasons. Thereafter, we take a different approach to the multi-query problem, with the first version of the algorithm running in exponential time, which is subsequently reduced to polynomial using quadratically constrained quadratic programming (QCQP). We show that this algorithm achieves an error of

$$O\left(\sqrt[2]{d} \cdot DS_Q(I) \cdot (\sqrt[2]{\log(1/\delta) + \log (DS_Q(I))}/\epsilon^2)\right),$$

matching the lower bound up to polylogarithmic factors.

Finally, we built a system prototype that can accept a set of SJA queries consisting of arbitrary joins, selection predicates, followed by aggregation. It can also automatically rewrites a group-by query into such a set of SJA queries and answer them with our query-answering mechanism. Experimental results demonstrate that our mechanism can significantly outperform privacy composition combined with the state-of-the-art single-query mechanism \cite{10}, especially on more skewed data and large $d$. 


2 RELATED WORK

In the past several years, query answering under differential privacy [14] has attracted a lot of attention [3, 10, 11, 13, 22, 27, 30, 32, 35, 36, 40]. Early works did not consider FK constraints, or equivalently, they adopt a DP model that only protects the tuples, which is called tuple-DP. Starting from [27], people began to consider user-DP modeled by FK constraints. Under user-DP, self-join-free queries [27, 40] are actually equivalent to the sum (mean) estimation problem [2, 12, 21]. Very recently, R2T [10] is proposed to deal with queries with self-joins while achieving instance optimality.

There are also a number of works studying graph pattern counting queries under differential privacy [7, 9, 24, 26, 34, 44], which is an important special case of SJA queries. For graph data, there are two DP policies: edge-DP [7, 24, 34, 44] and node-DP [7, 9, 26]. They correspond to tuple-DP and user-DP applied to the special schema $\mathcal{R} = \{\text{Node}(\text{ID}), \text{Edge}(\text{src}, \text{dst})\}$, respectively.

All the aforementioned works answer a single query at a time. The multi-query problem has been studied extensively on a flat table under tuple-DP [4, 6, 15, 19, 20, 29, 31, 33, 37–39, 41–43, 45]. For a set of $d$ arbitrary linear queries, advanced composition or the $d$-dimensional Gaussian mechanism achieves $\hat{O}(\sqrt{d})$ error for each query, which is the best we can achieve for $d < n$, where $n$ is the size of the table. For $d > n$, the optimal error of each query is $\hat{O}(\sqrt{n})$ [19]. For a set of queries with special structures, the error can be further reduced [15, 31, 41, 44]. Furthermore, [6, 29, 33] design mechanisms that are optimal for any given query set.

3 PRELIMINARIES

3.1 Database with FK Constraints

We first review the DP definition in databases with FKs [27]. Let $\mathcal{R}$ be the database schema. When foreign-key constraints exist, $\mathcal{R}$ can be formalized as a directed acyclic graph, where each node corresponds to a relation $R \in \mathcal{R}$ and a directed edge from $R' \rightarrow R$ indicates an FK reference from an attribute of $R'$ to the PK of $R$. One relation is designated as the primary private relations $R_P$ (this is without loss of generality; see example below), while a relation having a direct or indirect FK referencing $R_P$ is called a secondary private relation; relations having no FK references to $R_P$ are public.

Example 3.1. Consider the TPC-H schema in Figure 1: Suppose we want to protect the privacy of both the customers and the suppliers. Then we can add a virtual relation User(ID), which includes all the PKs in Customer and Supplier, while adding FK constraints from the PKs of Customer, Supplier to 1D. Then, User becomes the only primary private relation, while Customer, Supplier, Lineitem, Order, Part supp are secondary private relations, and Part, Nation, Region are public. □

Let $I$ be a database instance over $\mathcal{R}$. For any $R \in \mathcal{R}$, let $I(R)$ be the relation instance of $R$ in $I$. The reference relationships over tuples are defined naturally as follows: For $t \in I(R)$ and $t' \in I(R')$, we say $t'$ references $t$ if (1) $R'$ references $R$ and the FK of $t'$ is same as the PK of $t$; or (2) there exists another $t''$ such that $t'$ references $t''$ and $t''$ references $t$. Let $N = |I(R_P)|$ be the number of private users. For $i \in [N]$, let $t_i(I)$ be the $i$th user in $I(R_P)$. We write $I' \subseteq I$ if $I'(R_P) \subseteq I(R_P)$, and for each tuple $t$ in a secondary relation, it is included in $I'$ iff all tuples in $I(R_P)$ that are referenced by $t$ are included in $I'(R_P)$. This way, it generalizes the notion of an "induced subgraph" and we can call $I'$ an induced sub-instance of $I$. Note that an induced sub-instance $I'$ is completely specified by $I'(R_P)$. Besides, let $[n] = \{1, 2, \ldots, n\}$.

3.2 SJA Queries

We now define the class of queries considered. We start with a multi-way (natural) join:

$$J := R_1(x_1) \bowtie \cdots \bowtie R_n(x_n),$$

where $R_1, \ldots, R_n$ are relation names in $\mathcal{R}$ and each $x_i$ is a set of arity($R_i$) variables. Let $x := x_1 \cup \cdots \cup x_n$. When self-joins are present, there can be repeats among $R_1, \ldots, R_n$, i.e., $R_i = R_j$ (then we must have $x_i \neq x_j$). The join is required to be complete [27], i.e., if any $R_i$ in $J$ references another $R'$, then $R'$ must also be included in $J$, with its PK given a variable that is the same as the variable given to the corresponding FK of $R_i$.

Let $M = |J(I)|$ be the join size, let for $j \in [M]$, $q_j(I)$ denotes the $j$th join result. We say $q_j(I)$ references $t_i(I)$ iff $q_j(I) \bowtie t_i(I) \neq \emptyset$. To better describe the referencing relationships between tuples and join results, we introduce the following sets of indices $C_I(I)$ and $D_I(I)$ for each $i \in [N], j \in [M]$:

$$C_I(I) := \{j : q_j(I) \text{ references } t_i(I)\},$$

$$D_I(I) := \{i : q_j(I) \text{ references } t_i(I)\}.$$  

A JA or SJA query $Q$ aggregates over the join results $J(I)$. They can both be captured by introducing a function $\psi : \text{dom}(x) \to \mathbb{N}$ and defining the query result on $I$ as

$$Q(I) := \sum_{j \in [M]} \psi(q_j(I)).$$

Note that if there is a selection predicate, we can simply set $\psi(q) = 0$ for any $q$ that does not satisfy the predicate. We also use the shorthand $\psi_j(I) := \psi(q_j(I))$.

Example 3.2. Continuing with Example 3.1, suppose we ask the following query:

SELECT count(*) FROM Lineitem WHERE Lineitem.QTY > 10
Although this query does not have any explicit joins, we must make it complete, resulting in the following join:

\[ J = \text{Orders(OK, CK) } \bowtie \text{Lineitem(OK, SK, PTK, QTY)} \bowtie \text{Customer(CK, NK)} \bowtie \text{User(CK)} \bowtie \text{PartSupp(PTK, SK)} \bowtie \text{Supplier(SK, NK)} \bowtie \text{User(OK)} \]

Note how it introduces a self-join on User. Generally speaking, self-joins will be introduced as long as a join result references more than one private user. In fact, the join with the two copies of User has no effect on the join results; their purpose is to just mark some lineitems belonging to the \(i\)-th user (a customer or a supplier), and \(D_j(I)\) consists of all the users that the \(j\)-th lineitem references (exactly one customer and one supplier in this case).

Example 3.3. Many graph pattern matching queries under node-DP [7, 9, 26] can be written as SJIA queries on the schema \(R = (\text{Node}(\text{ID}), \text{Edge}((\text{src}, \text{dst})))\), where \(\text{src}, \text{dst}\) are both FKS referencing \(\text{ID}\). For example, the triangle counting query uses the join \(\text{Edge}(A, B) \bowtie \text{Edge}(B, C) \bowtie \text{Edge}(C, A) \bowtie \text{Node}(A) \bowtie \text{Node}(B) \bowtie \text{Node}(C)\) with \(\psi(q) = 1\) for all \(q \in J(I)\). Note that the 3 copies of Node are not really needed, but they are included (virtually to signify) that \(A, B, C\) all represent private users. For this query, \(C_j(I)\) includes all the triangles incident to the \(i\)-th node, and \(D_j(I)\) is the 3 nodes forming the \(j\)-th triangle.

In this paper, we consider answering \(d\) such queries \(Q = (Q_1, \ldots, Q_d)\). We subscript them by \(q\) and \(Q\), \(Q = (Q_1, \ldots, Q_d)\), \(Q = (Q_1, \ldots, Q_d)\), \(Q = (Q_1, \ldots, Q_d)\), while \(\gamma\) and \(\Gamma\) will be used for \(Q_1\) and \(Q_2\), and \(\gamma\) and \(\Gamma\) for \(Q_3\) and \(Q_4\).

#### 3.3 Differential Privacy

Definition 3.4 (Differential privacy). For \(\epsilon, \delta > 0\), an algorithm \(M: I \rightarrow \mathcal{Y}\) is \((\epsilon, \delta)\)-differentially private if for any neighboring instances \(I \sim I' \in I\) and any subset of outputs \(Y \subseteq \mathcal{Y}\),

\[ \Pr[M(I) \in Y] \leq e^\epsilon \cdot \Pr[M(I') \in Y] + \delta. \]

The privacy parameter \(\epsilon\) is usually a constant from 0.1 to 10, while \(\delta \ll 1/N\). To apply the DP definition to a concrete problem, we need to define the neighboring relationship. We adopt the user-DP policy in [27] for relational database, which defines two instances \(I\) and \(I'\) as neighbors if \(I'\) can be obtained from \(I\) by deleting some tuples \(T \in I(R_p)\) and all tuples referencing \(T\).

Lemma 3.5 (Laplace Mechanism [14]). Given a single query \(Q: I \rightarrow \mathbb{R}\), let \(G_{Q} = \max_{I'} |Q(I) - Q(I')|\). The mechanism

\[ M(I) = Q(I) + \frac{G_{Q}}{\epsilon} \cdot \mathcal{L}(1), \]

where \(Y \sim \text{Lap}(1), \) preserves \((\epsilon, 0)\)-DP.

For a vectored-valued query, the Gaussian mechanism is more commonly used:

**Lemma 3.6 (Gaussian Mechanism [8, 14]).** Given a \(d\)-dimensional query \(Q : I \rightarrow \mathbb{R}^d\), let \(G_{Q} = \max_{I'} |Q(I) - Q(I')|\). Then the mechanism

\[ M(I) = Q(I) + \sigma \cdot \mathcal{N}(0, I_d), \]

where \(Y \sim \mathcal{N}(0, I_d)\), preserves \((\epsilon, \delta)\)-DP for any \(\delta > 0\).

Thus, given \(\epsilon, \delta\), one can set \(\sigma = \sqrt{\frac{G_{Q}}{\epsilon} + \sqrt{2 \log(1/\delta)} \cdot \sqrt{d}}\) and solve for \(\sigma\), and we denote the solution as \(\sigma(\epsilon, \delta)\). For \(\epsilon = O(1)\), \(\sigma(\epsilon, \delta) = O\left(\sqrt{\log(1/\delta)}/\epsilon \cdot G_{Q}\right)\), so the Gaussian mechanism achieves an \(\epsilon\) error of \(\mathcal{O}\left(\sqrt{d} \cdot G_{Q}\right)\).

Both the Laplace and the Gaussian mechanism require a small GS. If it is large or unbounded, one may consider the local sensitivity \(LS(I) = \max_{I'} |Q(I) - Q(I')|\), and try to obtain a DP upper bound of \(LS(I)\) before applying the mechanism. Karwa et al. [25] show how this can be done for the Laplace mechanism. Here we obtain a similar result for the Gaussian mechanism:

**Lemma 3.7 (Second-order Gaussian Mechanism).** Given a \(d\)-dimensional query \(Q : I \rightarrow \mathbb{R}^d\), suppose there is an \((\epsilon_1, \delta_1)\)-DP mechanism that outputs an \(LS(I) \geq LS(I)\) with probability at least \(1 - \delta_2\), then the mechanism

\[ M(I) = Q(I) + \mathcal{N}(0, I_d), \sigma(\epsilon_2, \delta_2) \cdot Y, \]

where \(Y \sim \mathcal{N}(0, I_d)\), preserves \((\epsilon_1 + \epsilon_2, \delta_1 + \epsilon_1 \delta_2 + \epsilon_1 \delta_3)\)-DP.

Proof. Similar to the proof of Lemma 4.4 in [25].

However, in a relational database, \(LS(I)\) is also unbounded, since one can construct an \(I'\) by adding to a user \(T\) with arbitrarily many tuples all referencing \(T\). Thus, Dong et al. [10] propose to consider the downward sensitivity:

\[ DS(I) = \max_{I' \subseteq I} |Q(I') - Q(I)|. \]

Note by adding the restriction \(I' \subseteq I\), \(DS(I)\) corresponds to the maximum contribution of any user in \(I\), which is always finite. Specifically, for each user \(t_i(I) \in I(R_p)\), let

\[ S_{k,i}(I) = \sum_{j \in C_{k,i}(I)} \psi_{k,i}(j) \]

be the contribution of \(t_i(I)\) to \(Q_k\). The contributions of \(t_i(I)\) to all queries are thus a \(d\)-dimensional vector \(S_i(I) = (S_{1,i}(I), \ldots, S_{d,i}(I))\). Then we have \(DS(I) = \max_{e \in [N]} \|S_e(I)\|\).

#### 3.4 Sparse Vector Technique

The Sparse Vector Technique (SVT) [16] has as input a (possibly infinite) sequence of 1-dimensional queries, \(f_1(I), f_2(I), \ldots\), where each has global sensitivity 1, and a threshold \(T\). It targets to find the first query whose answer is above \(T\). SVT has been shown to satisfy \(\epsilon\)-DP with the following utility guarantee:

**Lemma 3.8 ([12, 16]).** If there exists a \(k\) such that \(f_k(D) \geq T + \frac{\delta}{2} \log(2/\beta)\), then with probability at least \(1 - \beta\), SVT returns an \(t \leq k\) such that \(f_t(D) \geq T - \frac{\delta}{2} \log(2k/\beta)\).
4 MULTIPLE SELF-JOIN-FREE QUERIES

We start with the simpler case of $d$ self-join-free queries. We observe that it is equivalent to the sum estimation problem in $d$ dimensions: Given $N$ vectors in $d$ dimensions $x_1, \ldots, x_N$, we wish to estimate $\sum_{i \in [N]} x_i$ under DP where neighboring instances differ by one vector. For the forward direction, we just set $x_i := S_i$. Since there is no self-join, each join result only references one user, then adding/removing one user in $I$ is the same as adding/removing vector. For the backward direction, we simply construct a single table with $d$ columns storing these vectors, and the $k$-th query asks for the sum on the $k$-th column.

This equivalence has two immediate consequences. First, the lower bound on the sum estimation problem is also a lower bound for the multi-query problem, which of course holds also for the more difficult case of self-joins:

**Theorem 4.1 ([21, 23]).** For the multi-query problem, no DP mechanism can achieve an error smaller than $\tilde{O}\left(\sqrt{d} \cdot DS_Q(I)\right)$ for all $I$.

Secondly, any sum estimation mechanism can also be used for self-join-free queries. However, all existing mechanisms assume that $||S_i|| \leq GS_Q$ for a predefined $GS_Q$. Under this assumption, the best result is [21]2

$$O\left(DS_Q(I) \cdot \left(\sqrt{d} + \log(GS_Q)\sqrt{\log(GS_Q)}\right) \cdot \sqrt{\log(1/\delta)/\epsilon}\right).$$

Below, we show how to remove this assumption, i.e., we allow $GS_Q = \infty$. Meanwhile, we also improve the error bound to

$$O\left(DS_Q(I) \cdot \left(\sqrt{d} \log(1/\delta) + \log(GS_Q)\sqrt{\log(GS_Q)}\right) / \epsilon\right).$$

Our idea is to extend the 1-dimensional mechanism in [12] to $d$ dimensions. Given a truncation threshold $r \geq 0$, the truncated query result $Q(I, r)$ is defined as

$$Q(I, r) := \sum_{i \in [N]} \left(\min\left(1, \frac{r}{||S_i||}\right) \cdot S_i\right).$$

It is easy to see that $Q(I, r)$ has the global sensitivity $r$, so the Gaussian mechanism can be applied. Note that by using $r = DS_Q(I)$, no data is truncated and the Gaussian mechanism achieves the optimal error $\tilde{O}\left(\sqrt{d} \cdot DS_Q(I)\right)$. However, since $DS_Q(I)$ depends on $I$, using it directly will breach privacy. Then the idea is to find a privatized $r$ that is as close to $DS_Q(I) = \max_{i \in [N]} ||S_i||$ as possible.

For any $r \geq 0$, define

$$\text{Count}(I, r) = |\{i : ||S_i|| \leq r\}|.$$

Then consider the sequence of queries $\text{Count}(I, r) - N$ for $r = 1, 2, 4, \ldots$. It is clear that each such query has global sensitivity 1. We use SVT with privacy budget $\epsilon/10$ to find the first $\tilde{r}$ such that $\text{Count}(I, \tilde{r}) - N \geq -\frac{\epsilon}{10} \log \frac{4}{\epsilon}$, where $\beta$ will be failure probability. After finding such an $\tilde{r}$, we invoke the Gaussian mechanism with global sensitivity $\tilde{r}$ and privacy budget $9\epsilon/10$.3 The detailed algorithm is shown in Algorithm 1.

---

Algorithm 1: Multiple self-join-free queries.

**Input:** $I, \epsilon, \beta$

1. $i \leftarrow \text{SVT}(\frac{-\epsilon}{10} \log \frac{4}{\epsilon}, \text{Count}(I, z) - N, \text{Count}(I, 2^i) - N, \ldots)$;
2. $\tilde{r} \leftarrow 2^{i-1}$;
3. $\tilde{Q}(I) = \tilde{Q}(I, \tilde{r}) + r \cdot \sigma_{\frac{\epsilon}{10}, \delta} \cdot Y, Y \sim N(0, I_{d \times d})$;
4. return $\tilde{Q}(I)$.

---

**Theorem 4.2.** Algorithm 1 satisfies $(\epsilon, \delta)$-DP, and with probability at least $1 - \beta$,

$$\|\tilde{Q}(I) - Q(I)\| = O\left(\frac{DS_Q(I)}{\epsilon} \cdot \left(\log(GS_Q) + \sqrt{d \log(1/\delta) \log(1/\beta)}\right)\right).$$

**Proof.** The privacy guarantee follows directly from basic composition. For the utility, by Lemma 3.8, with probability at least $1 - \frac{\beta}{7}$, we have (1) $\text{Count}(I, \tilde{r}) \geq N - O\left(\frac{1}{\epsilon} \log DS_Q(I)\right)$; and (2) $\tilde{r} \leq DS_Q(I)$. The first result implies the introduced bias is $O\left(\frac{DS_Q(I)}{\epsilon} \cdot \log DS_Q(I)\right)$. By combining the second result and the tail bound of Gaussian distribution, with probability at least $1 - \frac{\beta}{7}$, the added noise is $O\left(\frac{DS_Q(I)}{\epsilon} \cdot \sqrt{d \log(1/\delta) \log(1/\beta)}\right)$. □

5 MULTIPLE QUERIES WITH SELF-JOINS

5.1 Why Self-joins are Hard

When self-joins are present, or more fundamentally, when a join result references more than one user, a number of difficulties arise. First, since the contributions of the users overlap, the problem is no longer equivalent to sum estimation. Second, as pointed out in [10], the truncation mechanism fails: The query on all users $i$ with $S_i \leq r$ still has unbounded global sensitivity. To address this issue, [10] replaces this “hard” truncation with a “soft” truncation [26] for the case of a single query. The idea is that each join result may contribute a part of its full $S_i$, so that the total contribution from any user is bounded by $r$. This can be formulated as an LP, where the variable $z_j$ denotes the partial contribution from the $j$-th join result:

$$\max Q(I, r) = \sum_{j \in [M]} z_j$$

s.t. $\sum_{j \in C_i(I)} z_j \leq r, \quad i \in [N]$,

$$0 \leq z_j \leq \psi_j(I), \quad j \in [M].$$

The truncated query answer, denoted $Q(I, r)$, is set to be the optimal solution of this LP, which can be shown to have global sensitivity $r$, so the Laplace mechanism can be applied. Then, [10] further proposes a mechanism to privately select an $r$ to achieve the optimal error $\tilde{O}(DS_Q(I))$.

This LP can be naturally extended to multiple queries: $z_j$ and $\psi_j(I)$ both become $d$-dimensional vectors, the first constraint imposes a bound $r$ on the $\ell_2$ norm of $\sum z_j$, and the second constraint becomes an element-wise inequality. Meanwhile, $Q(I, r) = \sum z_j$ also becomes a vector, and we may try to maximize its norm. This turns the LP in a quadratic program (QP), which is still efficiently
solvable. But the critical issue is that $Q(I, r)$ has high local sensitivity, as illustrated in the following example.

![Diagram](image)

**Figure 2: An example showing $Q(I, r)$ has large sensitivity.**

**Example 5.1.** Consider the query in Example 3.2 with GROUP BY `OrderDate`. Suppose there are only two different dates on `OrderDate`, so we have just $d = 2$ queries. Consider an instance $I$ that has $N/2$ suppliers $s_1, s_2, \ldots, s_{N/2}$ and $N/2$ customers $c_1, c_2, \ldots, c_{N/2}$. Each lineitem is purchased by one customer and supplied by a supplier, and the detailed construction is shown in Figure 2, where solid lines denote the lineitems on day 1 and dashed lines for day 2. Next, we construct $I'$ by deleting $s_1$ (the associated lineitem) and construct $I''$ by further deleting $c_{N/2}$. Note that we have $I \sim I' \sim I''$.

Suppose we set $r = 1$. We see that $\|Q(I, 1)\|$ is maximized by keeping all lineitems on day 1, yielding $Q(I, 1) = (N/2, 0)$, while $\|Q(I', 1)\|$ is maximized by keeping all lineitems on day 2, yielding $Q(I', 1) = (0, N/2 - 1)$. We see that although $\|Q(I, 1)\| - \|Q(I', 1)\|$ is small, $\|Q(I, 1) - Q(I', 1)\|$ is large, which means that one of $\|Q(I, 1) - Q(I', 1)\|$ and $\|Q(I', 1) - Q(I'', 1)\|$ must be large. Fundamentally, the reason is that although the LP (or QP) has low sensitivity in its optimal value $\|Q(I, r)\|$, it does not necessarily imply a low sensitivity on the optimal vector solution $Q(I, r)$, except in one dimension. However, the Gaussian mechanism needs a low sensitivity on $Q(I, r)$, not $\|Q(I, r)\|$.

### 5.2 An Exponential-time Algorithm

To address the issue above, we take a different approach to defining $Q(I, r)$ so that it has bounded sensitivity. First, define

$$E(I, r) = \max_{I' \subseteq \text{DS}(I')} |I'(r)|,$$

i.e., $-F(I, r)$ is the minimum number of users that need to be removed so that the contribution from any user is at most $r$. The following lemma is obvious:

**Lemma 5.2.** For any $r$ and any $I$, $F(I, r) \leq 0$. If $r \geq \text{DS}(I)$, then $F(I, r) = 0$.

More importantly, we show that $F(I, r)$ has sensitivity 1:

**Lemma 5.3.** For any $r$ and any $I \sim I' \subseteq I$, we have

$$F(I', r) - 1 \leq F(I, r) \leq F(I', r).$$

**Proof.** Let $N' = |I'_1(R_P)|$ and $I(R_P) = I'(R_P) \cup t_N(I)$. On one hand, it is trivial to see for any $r \geq 0$, $E(I, r) \geq E(I', r)$ since any $I' \subseteq I$ also has $I' \subseteq I'$.

On the other hand, given any $r \geq 0$, let $I' = \arg \max_{I' \subseteq \text{DS}(I')} |I'(r)|$.

Then, we can construct a $I'$ from $I$ by deleting all tuples referencing $t_N(I)$. Then, $I' \subseteq I'$, $\text{DS}(I') \subseteq \text{DS}(I) \leq r$ and $|I'(r)| = |I'(R_P)| - 1$. And this further means, $E(I', r) \geq E(I, r) - 1$.

Finally, combining $N = N' + 1$, the lemma follows. □

Because $F(I, r)$ has sensitivity 1, we can feed $E(I, 1), E(I, 2), E(I, 4), \ldots$ into SVT with threshold $-O(\log(1/\beta)/r)$. Using a similar argument as for our self-join-free algorithm, we can show that this will return an $\tilde{r}$ that is close to $\text{DS}(I)$.

It turns out that $E(I, r)$ is not only useful for finding the truncation threshold $\tilde{r}$, it also yields a new definition of the truncated query answer $Q(I, r)$ with bounded sensitivity. For any $r \geq 0$, define

$$Q(I, r) := Q(I'(r)), \text{ where } I'(r) = \arg \max_{I' \subseteq \text{DS}(I')} |I'(r)|.$$  \hspace{1cm} (9)

**Note** that $I'(r)$ may not be unique. In this case, we use an arbitrary tie-breaker (e.g., the $I'$ with the lexicographically smallest user IDs) to fix an $I'(r)$.

**Example 5.4.** Following the Example 5.1 and assuming $N = 6c$ for some $c \in \mathbb{N}$, for $I$, when $r = 1$, we have $E(I, 1) = 2N/3$, $I'(r)$ is obtained by deleting $s_{2i-1}$’s and $c_{2i}$’s for all $i \in [N/6]$, and $Q(I, 1) = (N/6, N/6)$. When $r \geq 2$, $E(I, r) = N$, $I'(r) = I$, and $Q(I, r) = (N/2, N/2 - 1)$. Meanwhile, for $I''$, when $r = 1$, we have $E(I'', 1) = 2N/3 - 1$, $I''(r)$ is obtained by deleting $c_{2i-1}$’s for all $i \in [N/6]$ and $s_{2j+1}$’s for all $j \in [N/6 - 1]$, and $Q(I'', 1) = (N/6 - 1, N/6)$. When $r \geq 2$, $E(I'', r) = N - 2$, $I''(r) = I''$, and $Q(I'', r) = (N/2 - 2, N/2 - 1)$.

We bound the local sensitivity of $Q(I, r)$ as follows.

**Lemma 5.5.** Given $r \geq 0$, for any $I \sim I'$,

$$\|Q(I, r) - Q(I', r)\| \leq (-2F(I, r) + 2) \cdot r.$$  

**Proof.** Here, we first consider the case $I(R_P) = I'(R_P) \cup t_N(I)$. For convenience, let

$$I' = \arg \max_{I' \subseteq \text{DS}(I')} |I'(r)|,$$

$$I'' = \arg \max_{I' \subseteq \text{DS}(I')} |I''(r)|.$$  

And define $I_1'' = I'' \cap I'''$.
Algorithm 2: Exponential-time mechanism for multiple queries with self-joins.

**Input:** \( \varepsilon, \delta, \beta \)

1. \( \bar{r} \leftarrow \text{SVT} \left( -\frac{20}{7} \log(6/\beta), \frac{1}{\varepsilon}, F(I, L_0), F(I, L_2), \ldots \right) \).
2. \( r \leftarrow 2^{\bar{r}-1} \).
3. \( T \leftarrow -2F(I, r) + 2 \).
4. \( \tilde{T} \leftarrow T + \text{Laplace}(\frac{2}{\varepsilon}) + \frac{\beta}{2} \log(e^{3\varepsilon/5}/\delta) \).
5. \( \tilde{Q}(I) \leftarrow \tilde{Q}(I, r) + \tilde{T} \cdot \sigma \left( \frac{2}{\varepsilon}, \frac{\beta}{2e^2} \right) \cdot \tilde{T} \cdot Y \sim \mathcal{N}(0, I_{d\times d}) \).
6. return \( \tilde{Q}(I) \).

First, by definition,
\[
\begin{align*}
[I(R_p) \setminus I'(R_p)] \leq 1 & \Rightarrow |I(R_p) \setminus I'(R_p)| \leq -F(I', r) + 1. \\
[I'(R_p) \setminus I'(R_p)] \leq -F(I', r) & \Rightarrow |I'(R_p) \setminus I'(R_p)| \leq -F(I', r) + 1.
\end{align*}
\]
Further recall \( I' \subseteq I \), we have
\[
|I'(R_p) \setminus I'(R_p)| \leq -F(I', r) + 1.
\]
Recall \( \text{DS}_Q(I') \leq r \),
\[
\|Q(I') - Q(I'_0)\| \leq -F(I', r) + 1 \cdot r. \tag{10}
\]
Symmetrically, we have
\[
\|Q(I') - Q(I'_0)\| \leq -F(I', r) \cdot r. \tag{11}
\]
Combining Lemma 5.3, (10), (11), the claim follows.

Similarly, for the other case, where \( I'(R_p) = I(R_p) \cup \{ t_{N+1}(I') \} \), we have,
\[
\|Q(I') - Q(I'_0)\| \leq (\text{DS}(I) + 1) \cdot r.
\]
With a similar argument, the claim also follows. \( \square \)

Note that Lemma 5.5 does not imply a small sensitivity for all \( r \). In particular, if \( r \) is very small, \( -F(I, r) \) can be as large as \( N \), resulting in the same issue as in Example 5.1. However, the crucial difference here is that we will only use an \( \bar{r} \) returned by the SVT, which has \( -F(I, \bar{r}) = O(1) \) with high probability. We can further add a Laplace noise to it so that it becomes a high-probability DP upper bound on the local sensitivity, and then apply the second-order Gaussian mechanism. The detailed algorithm is shown in Algorithm 2.

**Theorem 5.6.** For any \( \varepsilon, \delta, \beta > 0 \), and any \( I \), Algorithm 2 satisfies \((\varepsilon, \delta)\)-DP and returns a \( \tilde{Q}(I) \) such that with probability at least \( 1 - \beta \),
\[
\|\tilde{Q}(I) - Q(I)\| = O \left( \frac{\sqrt{d \log(1/\beta)}}{\varepsilon^2} \cdot \text{DS}(I) \cdot \left( \log \frac{\log(\text{DS}(I))}{\beta} + \log(e^{\varepsilon}/\delta) \right) \right).
\]

**Proof.** Privacy: By Lemma 5.3, each \( F(I, \cdot) \) has the sensitivity 1 thus line 1 and line 4 consume \( \frac{1}{\varepsilon} \) and \( \frac{\beta}{2e^2} \) privacy budget respectively. By composition theory, \( \tilde{T} \) preserves \( \frac{\beta}{2e^2} \)-DP and by the tail bound of Laplace distribution, with probability at least \( 1 - \frac{\delta}{2e^2} \), \( \tilde{T} \geq T \). By Lemma 3.7, \( \tilde{Q}(I) \) preserves \((\varepsilon, \delta)\)-DP.

Below we prove the utility bound. First, by Lemma 3.8, with probability at least \( 1 - \frac{\beta}{2} \),
\[
-F(I, r) = O \left( \frac{1}{\varepsilon} \log \frac{\log(\text{DS}(I))}{\beta} \right), \tag{12}
\]
and
\[
r \leq 2 \cdot \text{DS}(I). \tag{13}
\]
Second, by (12) and the definitions of \( \text{DS}(I) \) and \( Q(I, r) \),
\[
\|Q(I) - Q(I, r)\| = O \left( \frac{\text{DS}(I)}{\varepsilon} \log \frac{\log(\text{DS}(I))}{\beta} \right). \tag{14}
\]
Third, by (12) and the tail bound of Gaussian distribution, we have with probability at least \( 1 - \frac{\beta}{2} \),
\[
\tilde{Q}(I) - Q(I) = O \left( \frac{\sqrt{d \log(1/\beta)}}{\varepsilon^2} \cdot \text{DS}(I) \cdot \log \frac{\log(\text{DS}(I))}{\beta} + \log(e^{\varepsilon}/\delta) \right). \tag{16}
\]
Finally, the theorem follows by combining (14) and (16). \( \square \)

### 5.3 A Polynomial-time Algorithm

In this section, we show how to reduce the running time of Algorithm 2 to polynomial without affecting its utility bound. The computational bottleneck is \( E(I, r) \). We borrow a popular technique from approximation algorithms: formulate \( E(I, r) \) as an integer program, and solve its relaxed version.

Observe that any \( I' \subseteq I \) is specified by the users in \( I'(R_p) \). We introduce a variable \( y_i \in \{0, 1\} \) to indicate whether the \( i \)-th user is included in \( I'(R_p) \). The objective is to maximize \( \sum_i y_i \) subject to \( \sum_i y_i = \beta \).

By combining (13), (15) and the tail bound of Gaussian distribution, we have with probability at least \( 1 - \frac{\beta}{2} \),
\[
\|\tilde{Q}(I) - Q(I')\| \leq O \left( \frac{\sqrt{d \log(1/\beta)}}{\varepsilon^2} \cdot \text{DS}(I) \cdot \log \frac{\log(\text{DS}(I))}{\beta} + \log(e^{\varepsilon}/\delta) \right). \tag{16}
\]
Therefore, $E(I, r)$ is the optimal solution of the following integer program:

$$\max \sum_{i \in [N]} y_i, \quad \text{s.t.} \quad z_{k,j} \geq \sum_{i \in D_{k,j}(I)} y_i - |D_{k,j}(I)| + 1, \quad k \in [d], j \in [M_k]$$

$$\sum_{k \in [d]} \left( \sum_{j \in C_{k,i}(I)} (\psi_{k,j}(I) \cdot z_{k,j}) \right)^2 \leq r^2, \quad i \in [N],$$

$$y_i \in \{0, 1\}, \quad i \in [N],$$

$$z_{k,j} \in \{0, 1\}, \quad k \in [d], j \in [M_k].$$

By relaxing the integral constraint to $y_i \in [0, 1], z_{k,j} \in [0, 1]$, this program turns into a QCQP. Note that only convex QCQPs can be solved efficiently, which is indeed the case for our QCQP, by observing that the quadratic constraint is positive semi-definite.

Let $(y'_i)_i, (z_{k,j}')_{k,j}$ be the optimal fractional solution of this QCQP, and let $E(I, r) = \sum_i y'_i$. In approximation algorithms, one would then try to round $(y'_i)_i, (z_{k,j}')_{k,j}$ into integers, and show that the rounded solution is not too far from $E(I, r)$. However, for the private query answering problem, there is no need to do the rounding as we will not return the join results anyway. Instead, we only need to return the privatized aggregated query answer.

On the other hand, we must show that $E(I, r)$ preserves the three important sensitivity properties of $E(I, r)$, namely, Lemma 5.2–5.5.

Letting $\tilde{E}(I, r) = E(I, r) - \|I(R_P)\|$. The first property is trivial:

**Lemma 5.7.** For any $r$ and any $I, \tilde{E}(I, r) \leq 0$. If $r \geq DS_Q(I)$, then $\tilde{E}(I, r) = 0$.

Now we prove that $\tilde{E}(I, r)$ also has sensitivity 1.

**Lemma 5.8.** For any $r$ and any $I \sim I', I' \subseteq I$, we have

$$\tilde{E}(I', r) - 1 \leq \tilde{E}(I, r) \leq \tilde{E}(I', r).$$

**Proof.** Let $I(R_P) = I'(R_Q) \cup t_N(I)$. Assume that the join results referencing $t_N(I)$ are put at the end of $J(I)$, i.e., for every $k \in [d], \ C_{k,i}(I) = \{M_k - |C_{k,i}(I)| + 1, \ldots, M_k - 1, M_k\}$.

Let $(y'_i)_i, (z_{k,j}')_{k,j}$ and $(y''_i)_i, (z''_{k,j})_{k,j}$ be the optimal fractional solutions of the QCQP on $I$ and $I'$, respectively. On one hand, from $(y'_i)_i, (z_{k,j}')_{k,j}$, we can construct a valid solution of the QCQP for $I$ by setting $y''_i = 0, z''_{k,j} = 0$ for any $k \in [d], j \in C_{k,N}(I)$. Thus, the optimal solution on $I$ can only be higher.

On the other hand, by removing $y''_i, z''_{k,j}$ for any $k \in [k], j \in C_{k,N}(I)$ for all $k \in [k]$, from $(y''_i)_i, (z''_{k,j})_{k,j}$, we obtain a valid solution of the QCQP on $I'$. On this solution, we have $\sum_{i \in [N]} y''_i = E(I, r) - y''_N \geq E(I, r) - 1$, which implies that $\tilde{E}(I', r) \geq \tilde{E}(I, r) - 1$.

Finally, combining with $N = N' + 1$, the lemma follows. □

Lastly, we show how the optimal fractional solution $(y'_i)_i, (z_{k,j}')_{k,j}$ also leads to a $\widetilde{Q}(I, r)$ with bounded local sensitivity as in Lemma 5.5. First, it is easy to see that there must exist an optimal solution in which the constraint on each $z_{k,j}'$ is tight, i.e.,

$$z_{k,j}' = \max \left(0, \sum_{i \in D_{k,j}(I)} y_i - |D_{k,j}(I)| + 1\right). \tag{17}$$

If not, we could lower $z_{k,j}'$ to make it tight without violating the quadratic constraint or changing the objective. If there are still multiple optimal fractional solutions, we pick one using an arbitrary tie-breaker. Then, we define $\widetilde{Q}(I, r)$ as the query answers using the optimal fractional solution, i.e.,

$$\widetilde{Q}(I, r) = Q(I') = \{Q_1(I'), Q_2(I'), \ldots, Q_d(I')\},$$

where $Q_k(I') = \sum_{j \in [M_k]} \left(z_{k,j}', \psi_{k,j}(I)\right), k \in [d]. \tag{18}$

**Example 5.9.** Following the Example 5.4, when $r = 1$, for both $I$ and $I'$, we have all $z_{k,j} = \sqrt{2}/2$, and all $y_i = \sqrt{2}/4 + 1/2$, thus $E(I, 1) = \sqrt{\sqrt{2}N/4} + N/2, \tilde{Q}(1, 1) = (\sqrt{\sqrt{2}N/4} + \sqrt{\sqrt{2}N/4} - \sqrt{2}/2), \tilde{E}(I', 1) = \sqrt{\sqrt{2}N/4} + N/2, \tilde{Q}(1', 1) = (\sqrt{\sqrt{2}N/4} - \sqrt{2}/2, \sqrt{\sqrt{2}N/4} - \sqrt{2}/2)$. When $r \geq 2$, for both $I$ and $I'$, we have all $z_{k,j} = 1$ and all $y_i = 1$, thus $E(I, r) = N, \tilde{Q}(1, 1) = (N/2, N/2 - 1), \tilde{E}(I', 1) = N - 2, \tilde{Q}(1', 1) = (N/2 - 2, N/2 - 1). \quad \square$

We now bound the local sensitivity of $\widetilde{Q}(I, r)$:

**Lemma 5.10.** Given any $r \geq 0$, for any $I \sim I'$,

$$\|\tilde{Q}(I, r) - \tilde{Q}(I', r)\| \leq (2\tilde{E}(I, r) + 2) \cdot r. \tag{19}$$

**Proof.** We consider the case $I(R_P) = I'(R_Q) \cup t_N(I)$ and the other case can be shown similarly as the proof of Lemma 5.5. And similarly as the proof of Lemma 5.8, we assume the join results corresponding to $t_N(I)$ are put at the end.

For $I$, we have $(y'_i)_i, (z_{k,j}')_{k,j}$ and $Q(I')$ constructed as $E(I, r)$, (17), and (18). For $I'$, we have $(y''_i)_i, (z''_{k,j})_{k,j}$ and $Q(I')$. To unify the size of $(y'_i)_i$ and $(y''_i)_i$, we append one zero at the end of $(y''_i)_i.

And we process similarly for $(z_{k,j}')_{k,j}$.

By definition of $\tilde{E}(I, r)$ and $(y'_i)_i$, we have

$$N - \sum_{i \in [N]} y'_i = -\tilde{E}(I, r).$$

And similarly, we have

$$N - \sum_{i \in [N]} y''_i = -\tilde{E}(I', r) + 1 \leq -\tilde{E}(I, r) + 2, \tag{20}$$

where the inequality is by Lemma 5.5. By this setting, we have

$$\|Q(I') - Q(I')\| = \left\|\sum_{k \in [d]} \left(\sum_{j \in [M_k]} (z_{k,j}' - z_{k,j}) \cdot \psi_{k,j}(I)\right)^2\right\|$$

$$\leq \sum_{k \in [d]} \left(\sum_{j \in [M_k]} (z_{k,j}' - z_{k,j}) \cdot \psi_{k,j}(I)\cdot I(z_{k,j}' < z_{k,j})\right)^2.$$
The first line is by $F_2$, except that $\sum_k \leq \sum_k$. Then, we have

$$\sum_k \leq \sum_k (1-y_k^*) \cdot z_{k,j} \cdot \psi_k(I) \cdot \mathbb{I}(z_{k,j} < z_{k,j}').$$

(22)

For any $k \in [d], j \in [M_k]$ such that $z_{k,j}^* < z_{k,j}'$, we have

$$z_{k,j}^* - z_{k,j}' \leq z_{k,j} - z_{k,j}' \leq z_{k,j}' - z_{k,j} \leq (1-z_{k,j}) \cdot z_{k,j}^* \leq \sum_i \{(1-y_i^*) \cdot z_{k,j}^* \}.\tag{23}$$

The second line is by (22). The fourth line is by the triangle inequality under $f_2$ distance metric. The last line is by (19).

Similarly, with (20), we can show,

$$\sum_k \left( \sum_{j \in [M_k]} (z_{k,j}^* - z_{k,j}') \cdot \psi_k(I) \cdot \mathbb{I}(z_{k,j}^* < z_{k,j}') \right)^2 \leq \sum_k \left( \sum_{j \in [M_k]} (1-y_j^*) \cdot z_{k,j}' \cdot \psi_k(I) \cdot \mathbb{I}(z_{k,j}^* < z_{k,j}') \right)^2 \leq \sum_{i \in [N]} (1-y_i^*) \cdot r \leq -\tilde{F}(I, r) \cdot r \tag{24}$$

Finally, combining (21), (23), and (24), the lemma follows.

Our polynomial-time algorithm is thus the same as Algorithm 2, except that $F(I, r)$ and $Q(I, r)$ are replaced by $\tilde{F}(I, r)$ and $\tilde{Q}(I, r)$, respectively. Below we show that this replacement does not affect its privacy or utility:

**Theorem 5.11.** For any $\epsilon, \delta, \beta > 0$ and any $l$, the polynomial-time version of Algorithm 2 preserves $(\epsilon, \delta)$-DP, and returns a $\tilde{Q}(I)$ such that with probability at least $1-\beta$,

$$\| \tilde{Q}(I) - Q(I) \| = O\left( \frac{d \log(1/\beta) \log(e^\epsilon/\delta)}{\epsilon^2} \cdot DS(I) \cdot \left( \log \frac{\log(DS)}{\beta} + \log(e^\epsilon/\delta) \right) \right).$$

**Proof.** The privacy analysis remains the same as in the proof of Theorem 5.6, since $\tilde{F}(I, r)$ and $\tilde{Q}(I, r)$ have the same sensitivity properties as $F(I, r)$ and $Q(I, r)$. Below we analyze the utility.

First, by Lemma 3.8 and 5.2, with probability at least $1 - \frac{\beta}{\gamma}$,

$$\tilde{F}(I, r) = O\left( \frac{1}{\epsilon} \log \frac{\log(DS)}{\beta} \right),$$

and $\tilde{r} \leq 2 \cdot DS(I)$.

By the difference between ExpPrivMultiSJA and PolyPrivMultiSJA, we only need to bound the bias

$$\| Q(I) - \tilde{Q}(I, r) \| = O\left( \frac{DS(I)}{\epsilon} \log \frac{\log(DS)}{\beta} \right).$$

(26)

(27)

Then, we increment $\{y_i^*_I\}$ to one vector, i.e. all elements equal to 1, and update the corresponding $\{z_{k,j}^*\}_{k,j}$ and $Q(I)$ iteratively. For convenience, we use the subscript to denote the iteration and let $\{y_i^*_I\}, \{z_{k,j}^*\}_{k,j}$ and, $Q(I)$ be the ones for iteration $0$, i.e. $\{y_i^{(0)}\}_I = \{y_i^*\}_I, \{z_{k,j}^{(0)}\}_{k,j} = \{z_{k,j}^*\}_{k,j}$, and

$$Q(I^{(0)}) = Q(I^*).$$

(28)

At iteration $i \in [N]$, we increment $y_i^{(i)}$ from $y_i^*$ to 1. Then, we set $\{z_{k,j}^{(i)}\}_{k,j}$ and $Q(I^{(i)})$ as (17) and (18). With setting, we have

$$\| Q(I^{(i)}) - Q(I^{(i-1)}) \| = \sum_{k=1}^d \left( \sum_{j=1}^{M_k} (z_{k,j}^{(i)} - z_{k,j}^{(i-1)}) \cdot \psi_k(I) \right)^2 \leq \sum_{k=1}^d \left( \sum_{j \in C_k(I)} (1-y_j^*) \cdot \psi_k(I) \right)^2 \leq (1-y_j^*) \cdot \sum_{k=1}^d \left( \sum_{j \in C_k(I)} \psi_k(I) \right)^2 \leq (1-y_j^*) \cdot DS_Q(I) \tag{29}$$

The third line is because, at $i$th iteration, for any $k \in [d], z_{k,j}^{(i)}$ will not be updated if $j \notin C_k(I)$. The fourth line is by (17). The last line is by the definition of $DS_Q(I)$.

After all iterations, we have all $y_i^{(N)} = 1$ thus

$$Q(I^{(N)}) = Q(I).$$

(30)
Summing up (29) for all $i$ and combining (28) and (30), we get (26). Finally, with a similar process as the proof of Theorem 5.6, the claim follows. \qed

6 SYSTEM IMPLEMENTATION

Our algorithm can be implemented on top of any SQL engine and a QCQP solver. For our system prototype, we use PostgreSQL and MOSEK.

The first step is to extract $\{C_k, I_k\}_k$, $\{D_k, I_k\}_k$ from the join results. Note that the original query does not output this information, so the first step is to rewrite the query so that it also includes the PKs of the private entities, as illustrated in the following example.

Example 6.1. Consider Q5 of TPC-H benchmark:

```sql
SELECT nation_name, SUM(price * (1 - discount))
FROM Supplier, Lineitem, Orders, Customer, Nation
WHERE ... GROUP BY nation_name
```

We rewrite it as

```sql
SELECT nation_name, Supplier.SK, Customer.CK,
price * (1 - discount)
FROM Supplier, Lineitem, Orders, Customer, Nation
WHERE ...
```

From the results of the rewritten query, we then construct a series of QCQPs and feed them into the SVT. Note that these QCQPs only differ in the value of $r$. The SVT returns an $\bar{f}$, $\bar{F}(I, \bar{r})$, and the corresponding optimal fractional solution, from which we construct $\hat{Q}(I, \bar{r})$. Finally, we add Gaussian noise to $\hat{Q}(I, \bar{r})$.

Algorithm 3: SVT with jump start

Input: $T$, $\epsilon$, $k$, and a sequence of sensitivity-1 queries $f_1(I), f_2(I), \ldots$

1. $\bar{T} \leftarrow T + \text{Lap}(2/\epsilon)$;
2. $\bar{F} = \infty$;
3. for $\ell \leftarrow k, k - 1, \ldots, 1$ do
   4. $\nu_\ell \leftarrow \text{Lap}(4/\epsilon)$;
   5. if $f' + \nu_\ell < \bar{T}$ then
      6. $f_\ell(I) \leftarrow f'$;
   7. else
      8. Compute $f_\ell(I)$;
      9. $f' \leftarrow f_\ell(I)$;
   10. end
4. end
5. for $\ell \leftarrow 1, 2, \ldots$ do
   6. if $\ell > k$ then
      7. $\nu_\ell \leftarrow \text{Lap}(4/\epsilon)$;
      8. Compute $f_\ell(I)$;
   9. end
10. if $f_\ell(I) + \nu_\ell \geq \bar{T}$ then
   11. return $\ell$;
12. end

Optimizations. We observe that the computational bottleneck is to solve the series of QCQPs. To make them more efficient, we use the following two techniques. First, for each QCQP, we rewrite it into a conic programming, which is then solved by MOSEK with a homogeneous primal-dual algorithm. It turns out that solving the QCQP this way is much more efficient in practice.

The second technique reduces the number of QCQPs we have to solve. We observe that as we increase $r$, the optimal solutions of the QCQPs, hence the $\bar{F}(I, r)$'s, are monotonically increasing. Recall that SVT returns the first $r = 2^k$ such that $\bar{F}(I, 2^k) + \nu_\ell \geq \bar{T}$, where $\nu_\ell$ is a Laplace noise. Since the first few $\bar{F}(I, 2^\ell)$'s are unlikely to go above the threshold, our idea is to give a "jump start" to the SVT by skipping those QCQPs. Let $2^k$ be the smallest power of 2 no less than DS(1). By our analysis in the proof of Theorem 5.6, the SVT is likely to stop around $r = 2^k$. Our idea is then to first generate $\nu_\ell$ for all $\ell = 1, \ldots, k$ in advance, but only compute an $\bar{F}(I, 2^\ell)$ if it has a chance to be above $\bar{T}$ after adding $\nu_\ell$. By going backward from $2^k$ to 1 and exploiting the monotonicity of $\bar{F}(I, 2^\ell)$, this can eliminate many of them from having to be computed. The detailed algorithm is shown in Algorithm 3. Note that Algorithm 3 is nothing but a more efficient execution of the original SVT, so its privacy and utility guarantees remain the same. Furthermore, this technique can be applied to any SVT instantiation with a sequence of monotonic queries $f_1(I), f_2(I), \ldots$, as long as there is a good guess $k$ on the likely stopping position (note that $k$ can depend on private information), so we present Algorithm 3 in a more generic form.

7 EXPERIMENTS

In this section, we report our experimental results comparing our algorithm (denoted PMSQA) with state-of-the-art algorithms for answering SQA queries with group-by over both benchmark and real-world datasets. For self-join-free queries, we compare with OptMean [21]; for queries with (implicit) self-joins, we compare with R2T [10] combined with advanced composition [18].

7.1 Setup

Datasets. We use two types of datasets: TPC-H and Stack Overflow network dataset. The TPC-H schema has been discussed before and is shown in Figure 1. We use datasets with scale 0.125, 0.25, $\ldots$, 8. The default scale is 2, where there are about 15 million tuples.

The Stack Overflow network dataset is from SNAP [28] and records users’ interactions on the Stack Overflow website. Here, each node represents one user and the interactions are stored as edges with a timestamp. We use two graphs, corresponding to answer-to-question (a2q) and comment-to-answer (c2a), respectively. They contain 2,464,606 nodes, 17,823,525 edges, and 1,646,338 nodes, 25,405,374 edges, respectively. We have deleted the top 10% nodes with the highest degrees, as protecting their privacy would introduce too much error. The number of edges of stackoverflow – a2q and stackoverflow – c2a are then reduced to 1,468,092 and 1,425,352.

Queries. We use 8 queries over TPC-H schema. The first two are self-join-free queries while the others are self-join queries. The query structures are shown in Figure 3. Some of the queries are taken directly from TPC-H benchmark while others are designed
to test the algorithms under various settings, in particular different combinations of primary private relations and counting queries are used. Furthermore, for the same TPC-H query, we may use different group-by attributes. That is also to test algorithm under more settings: Different group-by attributes lead to different group sizes $d$ and different data distributions in groups. Another reason is that TPC-H queries have too many or too few groups, i.e., for TPC-H query $Q^7$, if adding predicates, there are only 8 groups while removing the predicates will lead to thousands of groups. Therefore, we removed the predicates and used a subset of group-by attributes to get a good $d$. More precisely, both $Q_1$, $Q_2$ correspond to TPC-H query $Q^{10}$ where Customer is assigned as the primary private relation but use different group-by attributes. $Q_3$ corresponds to TPC-H query $Q^5$. $Q_4$ and $Q_5$ correspond to TPC-H query $Q^7$ but with different group-by attributes. All these three queries have Customer and Supplier as the primary private relations. Finally, $Q_6$, $Q_7$, and $Q_8$ provide more primary private relation combinations: {PartSupp, Customer} and {PartSupp, Orders}.

For the group-by attributes, we use date attribute or/and nation attribute. There are three cases. In the first case, we use the nation attribute and it has 25 groups. In the second case, we use the date, where we select 100 dates with each one corresponding to one group. We also conduct experiments with various groupings, which will be discussed later. Furthermore, we also use their combinations to make the group: we group the query results by nation and month. To avoid the heavy computations brought by the large group size, we only select two months and the group size here is 50. The group size for each query is shown in the head of Table 1.

In another dimension, $Q_2$, $Q_6$, $Q_7$ are counting queries while the others do the sum aggregation over extended price and discount attributes in Lineitem relation. The numbers shown are in thousands.

For graph pattern counting queries, we use edge counting query $Q_{1\ldots}$ length-2 path counting query $Q_{2\ldots}$, and length-3 path counting query $Q_{3\ldots}$. We take the groups on the time attribute on the first edge. Here, we have 10 groups and each group has several dates. Each group in $Q_{1\ldots}$, $Q_{2\ldots}$, and $Q_{3\ldots}$ has 40, 80, 220 dates respectively.

**Experimental parameters.** We conduct all experiments on a Linux server with a 24-core 48-thread 2.2GHz Intel Xeon CPU and 256GB memory. Each program is allowed to use at most 24 threads. We use $l_2$ metric in the report of query result and the error and we call one mechanism has utility and high utility if the relative error is below 50% and 30% respectively. Each experiment is repeated 20 times and we remove 4 largest errors and 4 smallest errors and report the average error for the rest 12 runs. For the privacy budgets, we use $\varepsilon = 2, 4, 8$ and the default value is set to 4. Compared with the work of answering single query [10, 11, 13, 22, 40], we use larger $\varepsilon$. That is because answering multiple queries is much more complex and we need larger $\varepsilon$ to guarantee the utility [1]. We set $\delta$ to 1e-7 and the failure probability $\beta$ to 0.1. Both OptMean and R2T require an $GSQ$ as the input parameter. For TPC-H queries, we set $GSQ$ to 1e6. For graph pattern counting queries, we set a degree upper bound of $\Delta = 1, 000, 000$ and set $GSQ$ as the maximum number of graph patterns containing any node, i.e., $GSQ_{1\ldots} = D$, $GSQ_{2\ldots} = D^2$, and $GSQ_{3\ldots} = D^3$.

### 7.2 Experimental Results

**Compare with state-of-the-art algorithms.** We conduct the experiments on the TPC-H dataset and graph data to compare PMSJA with OptMean and R2T and the results are shown in Table 1 and Table 2 where we report both error level (relative error) and running time. For self-join-free cases, OptMean already has very high utility while PMSJA further reduces this error by more than 50%. This matches our theoretical analyses: PMSJA improves OptMean by a log factor and both of them match the lower bound up to log factors. For efficiency, PMSJA uses a bit more time than OptMean but they are nearly at the same level. That is because, for self-join-free queries, they have very similar processes: extracting the relationship between users and join results first and then finding some threshold to do the clipping with sub-linear time.

Then, we talk about the results of 12 self-join queries and make a comparison between PMSJA with R2T. For the utility, PMSJA has high utility (relative error below 30%) in all 12 queries except the two $Q_{1\ldots}$, in which PMSJA still has the utility (relative error below 50%). Meanwhile, R2T only has utility in two $Q_{1\ldots}$ queries, where the group size is small, i.e., $d = 10$. More precisely, for TPC-H queries, the error level of R2T is 3.91x ~ 8.1x of PMSJA and a larger $d$ is more likely to lead to a larger gap. For the graph pattern counting queries where $d = 10$, that ratio is reduced to 1.91x ~ 2.53x. That confirms our theoretical analysis that PMSJA has $\sqrt{d}$
Table 1: Comparison between PMSJA and state-of-the-art algorithms (OptMean [21] for self-join-free queries and R2T [10] for self-join queries) on TPC-H queries with group-by operator. We use data scale equal to 2, $\varepsilon = 4$ and report the relative error.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>stackoverflow – a2q</th>
<th>stackoverflow – c2a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>$Q_1$</td>
<td>$Q_2$</td>
</tr>
<tr>
<td>Error(%)</td>
<td>Time(s)</td>
<td>Error(%)</td>
</tr>
<tr>
<td>PMSJA</td>
<td>48,000</td>
<td>0.735</td>
</tr>
<tr>
<td>R2T</td>
<td>5.56</td>
<td>27.3</td>
</tr>
</tbody>
</table>

Table 2: Comparison between PMSJA and R2T [10] on graph pattern counting queries with $d = 10$ on different networks. We use $\varepsilon = 4$ and report relative error.

Figure 4: Running times and error levels of PMSJA and R2T for $Q_5$ with different $d$ and $\varepsilon$. 

improvement over R2T and matches the lower bound up to log factors. For efficiency, as mentioned before, PMSJA solves several convex QCQP’s and needs much more running time than R2T, which only solves a number of LPs. However, as the experiments show, in more than half of the cases (8/12), the running time of PMSJA is not much larger than R2T (less than 8x), although it can be much longer in the worst case.

Number of queries. To further examine the effects of the change of number of queries $d$, we use $Q_1$ but more/fewer dates so we can have different $d$’s. More precisely, we run it with $d = 12, 25, 50, \ldots, 400$ on the TPC-H dataset with scale 2 and $\varepsilon$ is set to 2, 4, 8. We plot both error levels and running times in Figure 4, where we also plot the $\ell_2$ norm of the query result and its running time. For the error level, first, it is not surprising to see, PMSJA always has an error lower than R2T. As $d$ increases, the error of PMSJA decreases at the same rate as that of the query result: both of them increase with $\sqrt{d}$. Meanwhile, for R2T, its error level increases linearly with $d$ before it catches up with the query result (see the figure with $\varepsilon = 8$). One interesting finding is, when R2T’s catches up with the query result, it increases at the same rate as the query result and will not surpass that (see the figures with $\varepsilon = 2$ and $\varepsilon = 4$). That is because R2T will always return a noised value between 0 and the real query result thus its error is at most the query result. However, this does not really have any benefit since we have already lost all utility when the error level reaches the query result.
On the other hand, we see the running time of real query sublinearly increases with $d$. By contrast, R2T and PMSJA have linear and sup-linear speeds respectively. That is because R2T runs each query independently and for every single query, increasing $d$ will only affect the assigned $\varepsilon$, which just brings a minor effect on the running time thus the running time has a linear dependency on $d$. Meanwhile, for PMSJA, increasing $d$ will complicate the convex QCQP’s it solves thus leading to a super-linear effect on the running time.

**Scalability.** Lastly, we conduct the experiments to see the effects of data scale changes. We use TPC-H datasets with scale $0.125, 0.2, \ldots, 8$ and run $Q_3$ and $Q_7$ with $\varepsilon = 2, 4, 8$. The results are shown in Figure 5a and 5b.

First, the error level of DPSJA barely changes with the data scale. That is because theoretically, it only depends on $DS_Q(I)$, which does not change much by the scale of TPC-H data. On the other hand, the error level of R2T first increases with query result but will then stay at some level. That is because its error guarantee also depends on $DS_Q(I)$ and as mentioned before, its error is also bounded by the query result.

For efficiency, both query and R2T have running time that increases linearly with the data scale while the running time of PMSJA has a super-linear dependency on the data scale.

**ACKNOWLEDGMENTS**

This work has been supported by HKRGC under grants 16201819, 16205420, and 16205422. We would also like to thank the anonymous reviewers who have made valuable suggestions on improving the presentation of the paper.

Figure 5: Running times and error levels of PMSJA and R2T with different queries, data scales and $\varepsilon$. 
REFERENCES


