R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys

Wei Dong Juanru Fang Ke Yi
Hong Kong University of Science and Technology
{wdongac,jfangad,yike}@cse.ust.hk

Yuchao Tao Ashwin Machanavajjhala
Duke University
{yctao,ashwin}@cs.duke.edu

ABSTRACT

Answering SPJA queries under differential privacy (DP), including graph pattern counting under node-DP as an important special case, has received considerable attention in recent years. The dual challenge of foreign-key constraints and self-joins is particularly tricky to deal with, and no existing DP mechanisms can correctly handle both. For the special case of graph pattern counting under node-DP, the existing mechanisms are correct (i.e., satisfy DP), but they do not offer nontrivial utility guarantees or are very complicated and costly. In this paper, we propose the first DP mechanism for answering arbitrary SPJA queries in a database with foreign-key constraints. Meanwhile, it achieves a fairly strong notion of optimality, which can be considered as a small and natural relaxation of instance optimality. Finally, our mechanism is simple enough that it can be easily implemented on top of any RDBMS and an LP solver. Experimental results show that it offers order-of-magnitude improvements in terms of utility over existing techniques, even those specifically designed for graph pattern counting.

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; SPJA query; Foreign-key constraint

ACM Reference Format:
Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, & Ashwin Machanavajjhala. 2022. R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3517844

1 INTRODUCTION

Differential privacy (DP), already deployed by Apple [11], Google [16], Microsoft [12], and the US Census Bureau [26], has become the standard notion for private data release, due to its strong protection of individual information. Informally speaking, DP requires indistinguishability of the query results whether any particular individual's data is included or not in the database. The standard *Laplace mechanism* first finds GS_Q, the *global sensitivity*, of the query, i.e., how much the query result may change if an individual's data is added/removed from the database. Then it adds a Laplace noise calibrated accordingly to the query result to mask this difference. However, this mechanism runs into issues in a relational database, as illustrated in the following example.

Example 1.1. Consider a simple join-counting query

$$Q := |R_1(\underline{x_1}, \dots) \bowtie R_2(x_1, x_2, \dots)|.$$

Here, the underlined attribute $\underline{x_1}$ is the primary key (PK), while $R_2.x_1$ is a foreign key (FK) referencing $R_1.x_1$. For instance, R_1 may store customer information where x_1 is the customer ID and R_2 stores the orders the customers have placed. Then this query simply returns the total number of orders; more meaningful queries could be formed with some predicates, e.g., all customers from a certain region and/or orders in a certain category. Furthermore, suppose the customers, namely, the tuples in R_1, are the entities whose privacy we aim to protect.

What's the GS_Q for this query? It is, unfortunately, ∞. This is because a customer, theoretically, could have an unbounded number of orders, and adding such a customer to the database can cause an unbounded change in the query result. A simple fix is to assume a finite GS_Q, which can be justified in practice because we may never have a customer with, say, more than a million orders. However, as assuming such a GS_Q limits the allowable database instances, one tends to be conservative and sets a large GS_Q. This allows the Laplace mechanism to work, but adding noise of this scale clearly eliminates any utility of the released query answer. □

1.1 The Truncation Mechanism

The issue above was first identified by Kotsogiannis et al. [23], who also formalized the *DP policy for relational databases with FK constraints*. The essence of their model (a rigorous definition is given in Section 3) is that the individuals and their private data are stored in separate relations that are linked by FKs. This is perhaps the most crucial feature of the relational model, yet it causes a major difficulty in designing DP mechanisms as illustrated above. Their solution is the *truncation mechanism*, which simply deletes all customers with more than τ orders before applying the Laplace mechanism, for some threshold τ. After truncation, the query has sensitivity τ, so adding a noise of scale τ is sufficient.

Truncation is a special case of Lipshitz extensions and has been studied extensively for graph pattern counting queries [22] and machine learning [1]. A well-known issue for the truncation mechanism is the bias-variance trade-off: In one extreme $\tau = GS_Q$, it degenerates into the naive Laplace mechanism with a large noise

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517844

(i.e., large variance); in the other extreme $\tau = 0$, the truncation introduces a bias as large as the query answer. The issue of how to choose a near-optimal τ has been extensively studied in the statistics and machine learning community [2, 3, 18, 27, 33]. In fact, the particular query in Example 1.1 is equivalent to the 1-dimensional mean (sum) estimation problem, which is important for many machine learning tasks. A key challenge there is that the selection of τ must also be done in a DP manner.

1.2 The Issue with Self-joins

While self-join-free queries are equivalent to mean (sum) estimation (see Section 4 for a more formal statement), self-joins introduce another challenge unique to relational queries. In particular, all techniques from the statistics and machine learning literature [2, 3, 18, 27, 33] for choosing a τ critically rely on the fact that the individuals are independent, i.e., adding/removing one individual does not affect the data associated with another, which is not true when the query involves self-joins. In fact, when there are self-joins, even the truncation mechanism itself fails, as illustrated in the example below.

Example 1.2. Suppose we extend the query from Example 1.1 to the following one with a self-join:

$$Q := |R_1(\underline{x_1}, \dots,) \bowtie R_1(\underline{y_1}, \dots) \bowtie R_2(x_1, y_1, x_2, \dots)|.$$

Note that the PK of R_1 has been renamed differently in the two logical copies R_1, so that they join different attributes of R_2. For instance, R_2 may store the transactions between pairs of customers, and this query would count the total number of transactions. Again, predicates can be added to make the query more meaningful.

Let G be an undirected τ-regular graph (i.e., every vertex has degree τ) with n vertices. We will construct an instance $\mathsf{I} = (I_1, I_2)$ on which the truncation mechanism fails. Let I_1 be the vertices of G and let I_2 be the edges (each edge will appear twice as G is undirected). Thus Q simply returns the number of edges in the graph times 2. Let I' be the neighboring instance corresponding to G', to which we add a vertex v that connects to every existing vertex. Note that in G', v has degree n while every other vertex has degree $\tau + 1$. Now truncating by τ fails DP: The query answer on I is $n\tau$, and that on I' is 0 (all vertices are truncated). Adding noise of scale τ cannot mask this gap, violating the DP definition. □

The reason why the truncation mechanism fails is that the underlined claim above does not hold in the presence of self-joins. More fundamentally, this is due to the correlation among the individuals introduced by self-joins. In the example above, we see that the addition of one node may cause the degrees of many others to increase. For the problem of graph pattern counting under node-DP, which can be formulated as a multi-way self-join counting query on the special schema $\mathsf{R} = \{\texttt{Node}(\underline{\texttt{ID}}), \texttt{Edge}(\texttt{src}, \texttt{dst})\}$, Kasiviswanathan et al. [22] propose an LP-based truncation mechanism (to differentiate, we will call the truncation mechanism above *naive truncation*) to fix the issue, but they do not study how to choose τ. As a result, while their mechanism satisfies DP, there is no optimality guarantee in terms of utility. In fact, if τ is chosen inappropriately, their error can be even larger than GS_Q, namely, worse than the naive Laplace mechanism.

1.3 Our Contributions

In this paper, we start by studying how to choose a near-optimal τ in a DP manner in the presence of self-joins. As with all prior τ-selection mechanisms over mean (sum) estimation [2, 3, 18, 27, 33] and self-join-free queries [37], we assume that the *global sensitivity* of the given query Q is bounded by GS_Q. Since one tends to set a large GS_Q as argued in Example 1.1, we must try to minimize the dependency on GS_Q.

The first contribution of this paper (Section 5) is a simple and general DP mechanism, called *Race-to-the-Top (R2T)*, which can be used to adaptively choose τ in combination with any valid DP truncation mechanism that satisfies certain properties. In fact, it does not choose τ per se; instead, it directly returns a privatized query answer with error at most $O(\log(GS_Q) \log\log(GS_Q) \cdot DS_Q(\mathsf{I}))$ for any instance I with constant probability. While we defer the formal definition of $DS_Q(\mathsf{I})$ to Section 4, what we can show is that it is an *per-instance lower bound*, i.e., any valid DP mechanism has to incur error $\Omega(DS_Q(\mathsf{I}))$ on I (in a certain sense). Thus, the error of R2T is *instance-optimal* up to logarithmic factors in GS_Q. Furthermore, a logarithmic dependency on GS_Q is also unavoidable [20], even for the mean estimation problem, i.e., the simple self-join-free query in Example 1.1.

However, as we see in Example 1.2, naive truncation is not a valid DP mechanism in the presence of self-joins. As our second contribution (Section 6), we extend the LP-based mechanism of [22], which only works for graph pattern counting queries, to general queries on an arbitrary relational schema that uses the 4 basic relational operators: Selection (with arbitrary predicates), Projection, Join (including self-join), and sum Aggregation. When plugged into R2T, this yields the first DP mechanism for answering arbitrary SPJA queries in a database with FK constraints. For SJA queries, the utility is instance-optimal, while the optimality guarantee for SPJA queries (Section 7) is slightly weaker, but we argue that this is unavoidable.

Furthermore, the simplicity of our mechanism allows it to be built on top of any RDMBS and an LP solver. To demonstrate its practicality, we built a system prototype (Section 9) using PostgreSQL and CPLEX. Experimental results (Section 10) show that it can provide order-of-magnitude improvements in terms of utility over the state-of-the-art DP-SQL engines. We obtain similar improvements even over node-DP mechanisms that are specifically designed for graph pattern counting problems, which are just special SJA queries.

2 RELATED WORK

Answering arbitrary SQL queries under DP is the holy grail of private query processing. Most early work focuses on answering a given set counting queries over a single relation with different predicates (namely, SA queries with count aggregation) [6, 7, 10, 17, 25, 30, 35, 36, 40, 42]. [7, 25, 30] design mechanisms that are optimal for the given query set, but over the *worst-case* database. In particular, if the set consists of just one query, their optimality degenerates into worst-case optimality.

Most existing work on join queries can only support restricted types of joins, such as PK-PK joins [4, 28, 29, 32, 34] and joins with

a fixed join attribute [39]. A number of recent papers try to extend the support for joins, but as we see in Example 1.2, certain features like self-joins are tricky to handle correctly. PrivateSQL [23] uses naive truncation to truncate the tuples with high degree, so it does not really meet the DP requirement when there are self-joins. In a subsequent work, Tao et al. [37] use naive truncation to truncate the tuples with high sensitivity for self-join-free queries and they propose a mechanism to select τ. However, our analysis (see Appendix A) shows that the error of their mechanism is $\Omega(GS_Q/\log(GS_Q))$ with constant probability, i.e., it is at most a logarithmic-factor better than the naive Laplace mechanism that adds noise of scale GS_Q. We reduce the dependency on GS_Q from (near) linear to logarithmic. We also compare with their mechanism experimentally for self-join-free queries in Section 10.

Smooth sensitivity [31] is a popular approach for dealing with self-joins. Elastic sensitivity [19] and residual sensitivity [13, 14], both of which are efficiently computable versions of smooth sensitivity, can handle self-joins correctly. However, as we argue in Section 4, smooth sensitivity (including any efficiently computable version) cannot support FK constraints, which are important to modeling the relationship between an individual and all his/her associated data in a relational database. Consequently, they do not support node-DP for graph pattern counting, which is an important special case of FK constraints.

Node-DP and edge-DP are two popular DP policies for private graph data, which respectively are the special cases of having FK or no FK constraints in a relational database, as elaborated in Section 3.2. For node-DP, Kasiviswanathan et al. [22] combine naive truncation and smooth sensitivity, and also propose an LP-based truncation mechanism, while Blocki et al. [8] develop a smooth distance estimator. All of these mechanisms require a τ given in advance. As such, none of them has any utility guarantees. Experimentally, we show in Section 10 (cf. Table 3) that the error of these mechanisms is highly sensitive to τ, while there is no fixed τ that works well for all queries and datasets. On the other hand, R2T can always adaptively choose a τ that is provably close to the optimal one tuned (note that the tuning violates DP!) for each particular query/dataset. For edge-DP, better utility can be achieved [8, 21, 31, 41], but the privacy protection is weaker.

The recursive mechanism [9] also achieves an error close to $DS_Q(\mathsf{I})$, but without showing its instance optimality. More importantly, it adopts an approach that is complicated and different from the mainstream ones (e.g., the truncation mechanism and smooth sensitivity). In addition, its high computational costs prevent it from being used in practice. In our experiments, we were able to finish running this mechanism (with a time limit of 6 hours) only on the 3 test cases with the smallest query result size.

3 PRELIMINARIES

3.1 Database Queries

Let R be a database schema. We start with a multi-way join:

$$J := R_1(\mathsf{x}_1) \bowtie \cdots \bowtie R_n(\mathsf{x}_n), \tag{1}$$

where $R_1, \ldots, R_n$ are relation names in R and each x_i is a set of $arity(R_i)$ variables. When considering self-joins, there can be repeats, i.e., $R_i = R_j$; in this case, we must have $\mathsf{x}_i \neq \mathsf{x}_j$, or one of the two atoms will be redundant. Let $var(J) := \mathsf{x}_1 \cup \cdots \cup \mathsf{x}_n$.

Let I be a database instance over R. For any $R \in \mathsf{R}$, denote the corresponding relation instance in I as $\mathsf{I}(R)$. This is a *physical relation instance* of R. We use $\mathsf{I}(R, \mathsf{x})$ to denote $\mathsf{I}(R)$ after renaming its attributes to x, which is also called a *logical relation instance* of R. When there are self-joins, one physical relation instance may have multiple logical relation instances; they have the same rows but with different column (variable) names.

An JA or SJA query Q aggregates over the join results $J(\mathsf{I})$. More abstractly, let $\psi : \mathsf{dom}(var(J)) \to \mathsf{N}$ be a function that assigns non-negative integer weights to the join results, where $\mathsf{dom}(var(J))$ denotes the domain of $var(J)$. The result of evaluating Q on I is

$$Q(\mathsf{I}) := \bigotimes_{q \in J(\mathsf{I})} \psi(q). \tag{2}$$

Note that the function ψ only depends on the query. For a counting query, $\psi(\cdot) \equiv 1$; for an aggregation query, e.g. $\mathtt{SUM}(A*B)$, $\psi(q)$ is the value of $A * B$ for q. And an SJA query with an arbitrary predicate over $var(J)$ can be easily incorporated into this formulation: If some $q \in J(\mathsf{I})$ does not satisfy the predicate, we simply set $\psi(q) = 0$.

Example 3.1. Graph pattern counting queries can be formulated as SJA queries. Suppose we store a graph in a relational database by the schema $\mathsf{R} = \{\mathtt{Node}(\underline{\mathtt{ID}}), \mathtt{Edge}(\mathtt{src}, \mathtt{dst})\}$ where `src` and `dst` are FKs referencing `ID`, then the number of length-3 paths can be counted by first computing the join

$$\mathtt{Edge}(\mathtt{A}, \mathtt{B}) \bowtie \mathtt{Edge}(\mathtt{B}, \mathtt{C}) \bowtie \mathtt{Edge}(\mathtt{C}, \mathtt{D}),$$

followed by a count aggregation. Note that this also counts triangles and non-simple paths (e.g., x-y-x-z), which may or may not be considered as length-3 paths depending on the application. If not, they can be excluded by introducing a predicate (i.e., redefining ψ) $\mathtt{A} \neq \mathtt{C} \wedge \mathtt{A} \neq \mathtt{D} \wedge \mathtt{B} \neq \mathtt{D}$. If the graph is undirected, then the query counts every path twice, so we should divide the answer by 2. Alternatively, we may introduce the predicate $\mathtt{A} \ddot{Y} \mathtt{D}$ to eliminate the double counting. □

Finally, for an SPJA query where the output variables are $\mathsf{y} \subset var(J)$, we simply replace $J(\mathsf{I})$ with $\pi_{\mathsf{y}} J(\mathsf{I})$ in (2). Note that we use the relational algebra semantics of a projection, where duplicates are removed. If not, the projection would not make any difference in the aggregate. In fact, it is precisely the duplicate-removal that makes SPJA queries more difficult than SJA queries in terms of optimality, as we argue in Section 7.

3.2 Differential Privacy in Relational Databases with Foreign Key Constraints

We adopt the DP policy in [23], which defines neighboring instances by taking foreign key (FK) constraints into consideration. We model all the FK relationships as a directed acyclic graph over R by adding a directed edge from R to R' if R has an FK referencing the PK of R'. There is a[1] designated *primary private relation* R_P and any

[1]For most parts of the paper, we consider the case where there is only one *primary private relation* in R; the case with multiple primary private relations is discussed in Section 8.

relation that has a direct or indirect FK referencing R_P is called a *secondary private relation*. The *referencing* relationship over the tuples is defined recursively as follows: (1) any tuple $t_P \in \mathsf{I}(R_P)$ said to reference itself; (2) for $t_P \in \mathsf{I}(R_P)$, $t \in \mathsf{I}(R), t' \in \mathsf{I}(R')$, if t' references t_P, R has an FK referencing the PK of R', and the FK of t equals to the PK of t', then we say that t references t_P. Then two instances I and I' are considered neighbors if I' can be obtained from I by deleting a set of tuples, all of which reference the same tuple $t_P \in \mathsf{I}(R_P)$, or vice versa. In particular, t_P may also be deleted, in which case all tuples referencing t_P must be deleted in order to preserve the FK constraints. Finally, for a join result $q \in J(\mathsf{I})$, we say that q references $t_P \in \mathsf{I}(R_P)$ if $|t_P \bowtie q| = 1$.

We use the notation $\mathsf{I} \sim \mathsf{I}'$ to denote two neighboring instances and $\mathsf{I} \sim_{t_P} \mathsf{I}'$ denotes that all tuples in the difference between I and I' reference the tuple $t_P \in R_P$.

Example 3.2. Consider the TPC-H schema:

$$\mathsf{R} = \{\texttt{Nation}(\underline{\texttt{NK}}), \texttt{Customer}(\underline{\texttt{CK}}, \texttt{NK}), \texttt{Order}(\underline{\texttt{OK}}, \texttt{CK}), \texttt{Lineitem}(\texttt{OK})\}.$$

If the customers are the individuals whose privacy we wish to protect, then we designate `Customer` as the primary private relation, which implies that `Order` and `Lineitem` will be secondary private relations, while `Nation` will be public. Note that once `Customer` is designated as a primary private relation, the information in `Order` and `Lineitem` is also protected since the privacy induced by `Customer` is stronger than that induced by `Order` and `Lineitem`. Alternatively, one may designate `Order` as the primary private relation, which implies that `Lineitem` will be a secondary private relation, while `Customer` and `Nation` will be public. This would result in weaker privacy protection but offer higher utility. □

Some queries, as given, may be *incomplete*, i.e., it has a variable that is an FK but its referenced PK does not appear in the query Q. The query in Example 3.1 is such an example. Following [23], we always make the query complete by iteratively adding those relations whose PKs are referenced to Q. The PKs will be given variables names matching the FKs. For example, for the query in Example 3.1, we add `Node(A)`, `Node(B)`, `Node(C)`, and `Node(D)`.

The DP policy above incorporates both edge-DP and node-DP, two commonly used DP policies for private graph analysis, as special cases. In Example 3.1, by designating `Edge` as the private relation (`Node` is thus public, and we may even assume it contains all possible vertex IDs), we obtain edge-DP; for node-DP, we add FK constraints from `src` and `dst` to `ID`, and designate `Node` as the primary private relation, while `Edge` becomes a secondary private relation.

A mechanism M is ε-DP if for any neighboring instance I, I', and any output y, we have

$$\Pr[M(\mathsf{I}) = y] \le e^{\varepsilon} \Pr[M(\mathsf{I}') = y].$$

Typical values of ε used in practice range from 0.1 to 10, where a smaller value corresponds to stronger privacy protection.

4 INSTANCE OPTIMALITY OF DP MECHANISMS WITH FK CONSTRAINTS

Global sensitivity and worst-case optimality. The standard DP mechanism is the Laplace mechanism [15], which adds $Lap(GS_Q)$ to the query answer. Here, $Lap(b)$ denotes a random variable drawn from the Laplace distribution with scale b and $GS_Q = \max_{\mathsf{I}\sim\mathsf{I}'} |Q(\mathsf{I}) - Q(\mathsf{I}')|$ is the *global sensitivity* of Q. However, either a join or a sum aggregation makes GS_Q unbounded. The issue with the former is illustrated in Example 1.1, where a customer may have unbounded orders; a sum aggregation with an unbounded ψ results in the same situation. Thus, as with prior work [2, 3, 18, 27, 33, 37], we restrict to a set of instances $\mathcal{I}$ such that

$$\max_{\mathsf{I}\in\mathcal{I}, \mathsf{I}'\in\mathcal{I}, \mathsf{I}\sim\mathsf{I}'} |Q(\mathsf{I}) - Q(\mathsf{I}')| = GS_Q, \tag{3}$$

where GS_Q is a parameter given in advance. For the query in Example 1.1, this is equivalent to assuming that a customer is allowed to have at most GS_Q orders in any instance.

For general queries, the situation is more complicated. We first consider SJA queries. Given an instance I and an SJA query Q, for a tuple $t_P \in \mathsf{I}(R_P)$, its *sensitivity* is

$$S_Q(\mathsf{I}, t_P) := \bigotimes_{q\in J(\mathsf{I})} \psi(q)\mathbf{1}(q \text{ references } t_P), \tag{4}$$

where $\mathbf{1}(\cdot)$ is the indicator function. For SJA queries, (3) is equivalent to

$$\max_{\mathsf{I}\in\mathcal{I}} \max_{t_P\in\mathsf{I}(R_P)} S_Q(\mathsf{I}, t_P) = GS_Q.$$

For self-join-free SJA queries, it is clear that

$$Q(\mathsf{I}) = \bigotimes_{t_P\in R_P} S_Q(\mathsf{I}, t_P),$$

which turns the problem into a sum estimation problem. However, when self-joins are present, this equality no longer holds since one join result q references multiple t_P's. This also implies that removing one tuple from $\mathsf{I}(R_P)$ may affect multiple $S_Q(\mathsf{I}, t_P)$'s, making the neighboring relationship more complicated than in the sum estimation problem, where two neighboring instances differ by only one datum [2, 3, 18, 27, 33].

What notion of optimality shall we use for DP mechanisms over SJA queries? The traditional worst-case optimality is meaningless, since the naive Laplace mechanism that adds noise of scale GS_Q is already worst-case optimal, just by the definition of GS_Q. In fact, the basis of the entire line of work on the truncation mechanism and smooth sensitivity is the observation that typical instances should be much easier than the worst case, so these mechanisms all add instance-specific noises, which are often much smaller than the worst-case noise level GS_Q.

Instance optimality. The standard notion of optimality for measuring the performance of an algorithm on a per-instance basis is *instance optimality*. More precisely, let $\mathcal{M}$ be the class of DP mechanisms and let[2]

$$\mathcal{L}_{\text{ins}}(\mathsf{I}) := \min_{M'\in\mathcal{M}} \min\{\xi : \Pr[|M'(\mathsf{I}) - Q(\mathsf{I})| \le \xi] \ge 2/3\}$$

be the lower bound any $M' \in \mathcal{M}$ can achieve (with probability 2/3) on I, then the standard definition of instance optimality requires us to design an M such that

$$\Pr[|M(\mathsf{I}) - Q(\mathsf{I})| \le c \cdot \mathcal{L}_{\text{ins}}(\mathsf{I})] \ge 2/3 \tag{5}$$

for every I, where c is called the *optimality ratio*. Unfortunately, for any I, one can design a trivial $M'(\cdot) \equiv Q(\mathsf{I})$ that has 0 error on I

[2]The probability constant 2/3 can be changed to any constant larger than 1/2 without affecting the asymptotics.

(but fails miserably on other instances), so $\mathcal{L}_{\text{ins}}(\cdot) \equiv 0$, which rules out instance-optimal DP mechanisms by a standard argument [15].

To avoid such a trivial M', [5, 14] consider a relaxed version of instance optimality where we compare M against any M' that is required to work well not just on I, but also on its neighbors, i.e., we raise the target error from $\mathcal{L}_{\text{ins}}(\mathsf{I})$ to

$$\mathcal{L}_{\text{nbr}}(\mathsf{I}) := \min_{M' \in \mathcal{M}} \max_{\mathsf{I}' : \mathsf{I} \sim \mathsf{I}'} \min\{\xi : \Pr[|M'(\mathsf{I}') - Q(\mathsf{I}')| \le \xi] \ge 2/3\}.$$

Vadhan [38] observes that $\mathcal{L}_{\text{nbr}}(\mathsf{I}) \ge LS_Q(\mathsf{I})/2$, where

$$LS_Q(\mathsf{I}) := \max_{\mathsf{I}' \in \mathcal{I}, \mathsf{I}' \sim \mathsf{I}} |Q(\mathsf{I}) - Q(\mathsf{I}')|$$

is the *local sensitivity* of Q at I. This instance optimality has been used for certain machine learning problems [5] and conjunctive queries without FKs [14]. However, it has an issue for SJA queries in a database with FK constraints: For any I, we can add a t_P to $\mathsf{I}(R_P)$ together with tuples in the secondary private relations all referencing t_P, obtaining an I' such that $S_Q(\mathsf{I}', t_P) = GS_Q$, i.e., $LS_Q(\cdot) \equiv GS_Q$. This means that this relaxed instance optimality degenerates into worst-case optimality. This is also why smooth sensitivity, including all its efficiently computable versions [13, 14, 19, 31], will not have better utility than the naive Laplace mechanism on databases with FK constraints, since they are all no lower than the local sensitivity.

The reason why the above relaxation is "too much" is that we require M' to work well on any neighbor I' of I. Under the neighborhood definition with FK constraints, this means that I' can be any instance obtained from I by adding a tuple t_P and *arbitrary* tuples referencing t_P in the secondary private relations. This is too high a requirement for M', hence too low an optimality notion for M.

To address the issue, [18] restricts the neighborhood in which M' is required to work well, but their definition only works for the mean estimation problem. For SJA queries under FK constraints, we revise $\mathcal{L}_{\text{nbr}}(\cdot)$ to

$$\mathcal{L}_{\text{d-nbr}}(\mathsf{I}) := \min_{M' \in \mathcal{M}} \max_{\mathsf{I}' : \mathsf{I} \sim \mathsf{I}', \mathsf{I}' \subseteq \mathsf{I}} \min\{\xi : \Pr[|M'(\mathsf{I}') - Q(\mathsf{I}')| \le \xi] \ge 2/3\},$$

namely, we require M' to work well only on I' and its *down-neighbors*, which can be obtained only by removing a tuple t_P already in $\mathsf{I}(R_P)$ and all tuples referencing t_P. Correspondingly, an instance-optimal M (w.r.t. the down-neighborhood) is one such that (5) holds where $\mathcal{L}_{\text{ins}}$ is replaced by $\mathcal{L}_{\text{d-nbr}}$.

Clearly, the smaller the neighborhood, the stronger the optimality notion. Our instance optimality notion is thus stronger than those in [5, 14, 18]. Note that for such an instance-optimal M (by our definition), there still exist I, M' such that M' does better on I than M, but if this happens, M' must do worse on one of the down-neighbors of I, which is as typical as I itself.

Using the same argument from [38], we have $\mathcal{L}_{\text{d-nbr}}(\mathsf{I}) \ge DS_Q(\mathsf{I})/2$, where

$$DS_Q(\mathsf{I}) := \max_{\mathsf{I}', \mathsf{I} \sim \mathsf{I}, \mathsf{I}' \subseteq \mathsf{I}} |Q(\mathsf{I}) - Q(\mathsf{I}')| = \max_{t_P \in \mathsf{I}(R_P)} S_Q(\mathsf{I}, t_P) \tag{6}$$

is the *downward local sensitivity* of I. Thus, $DS_Q(\mathsf{I})$ is a per-instance lower bound, which can be used to replace $\mathcal{L}_{\text{inc}}(\mathsf{I})$ in (5) in the definition of instance-optimal DP mechanisms.

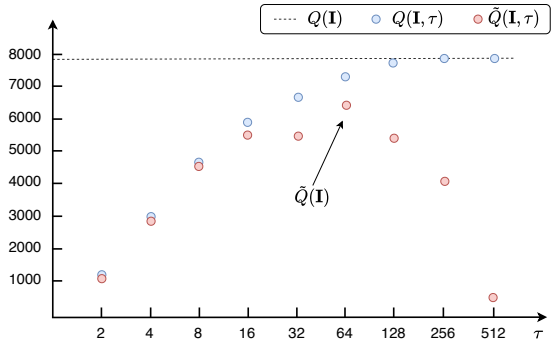

Figure 1: An illustration of R2T.

5 R2T: INSTANCE-OPTIMAL TRUNCATION

Our instance-optimal truncation mechanism, *Race-to-the-Top (R2T)*, can be used in combination with any truncation method $Q(\mathsf{I}, \tau)$, which is a function $Q : \mathcal{I} \times \mathsf{N} \to \mathsf{N}$ with the following properties:

(1) For any τ, the global sensitivity of $Q(\cdot, \tau)$ is at most τ.
(2) For any τ, $Q(\mathsf{I}, \tau) \le Q(\mathsf{I})$.
(3) For any I, there exists a non-negative integer $\tau^*(\mathsf{I}) \le GS_Q$ such that for any $\tau \ge \tau^*(\mathsf{I})$, $Q(\mathsf{I}, \tau) = Q(\mathsf{I})$.

We describe various choices for $Q(\mathsf{I}, \tau)$ depending on the DP policy and whether the query contains self-joins and/or projections in the subsequent sections. Intuitively, such a $Q(\mathsf{I}, \tau)$ gives a stable (property (1)) underestimate (property (2)) of $Q(\mathsf{I})$, while reaches $Q(\mathsf{I})$ for τ sufficiently large (property (3)). Note that $Q(\mathsf{I}, \tau)$ itself is not DP. To make it DP, we can add $Lap(\tau/\varepsilon)$, which would turn it into an ε-DP mechanism by property (1). The issue, of course, is how to set τ. The basic idea of R2T is to try geometrically increasing values of τ and somehow pick the "winner" of the race.

Assuming such a $Q(\mathsf{I}, \tau)$, R2T is works as follows. For a probability[3] β, we first compute[4]

$$\tilde{Q}(\mathsf{I}, \tau^{(j)}) := Q(\mathsf{I}, \tau^{(j)}) + Lap\left(\log(GS_Q)\frac{\tau^{(j)}}{\varepsilon}\right)^{!} - \log(GS_Q)\ln\left(\frac{\log(GS_Q)}{\beta}\right)\cdot\frac{\tau^{(j)}}{\varepsilon}, \tag{7}$$

for $\tau^{(j)} = 2^j$, $j = 1, \ldots, \log(GS_Q)$. Then R2T outputs

$$\tilde{Q}(\mathsf{I}) := \max\left(\max_j \tilde{Q}(\mathsf{I}, \tau^{(j)}), Q(\mathsf{I}, 0)\right). \tag{8}$$

The privacy of R2T is straightforward: Since $Q(\mathsf{I}, \tau^{(j)})$ has global sensitivity at most $\tau^{(j)}$, and the third term of (7) is independent of I, each $\tilde{Q}(\mathsf{I}, \tau^{(j)})$ satisfies $\frac{\varepsilon}{\log(GS_Q)}$-DP by the standard Laplace mechanism. Collectively, all the $\tilde{Q}(\mathsf{I}, \tau^{(j)})$'s satisfy ε-DP by the basic composition theorem [15]. Finally, returning the maximum preserves DP by the post-processing property of DP.

[3]The probability β only concerns about the utility but not privacy.
[4]log has base 2 and ln has base e.

Utility analysis. For some intuition on why R2T offers good utility, please see Figure 1. By property (2) and (3), as we increase τ, $Q(I,\tau)$ gradually approaches the true answer $Q(I)$ from below and reaches $Q(I,\tau) = Q(I)$ when $\tau \geq \tau^*(I)$. However, we cannot use $Q(I,\tau)$ or $\tau^*(I)$ directly as this would violate DP. Instead, we only get to see $\tilde{Q}(I,\tau)$, which is masked with the noise of scale proportional to τ. We thus face a dilemma, that the closer we get to $Q(I)$, the more uncertain we are about the estimate $\tilde{Q}(I,\tau)$. To get out of the dilemma, we shift $Q(I,\tau)$ down by an amount that equals to the scale of the noise (if ignoring the log log factor). This penalty for $\tilde{Q}(I,\hat{\tau})$, where $\hat{\tau}$ is the smallest power of 2 above $\tau^*(I)$, will be on the same order as $\tau^*(I)$, so it will not affect its error by more than a constant factor, while taking the maximum ensures that the winner is at least as good as $\tilde{Q}(I,\hat{\tau})$. Meanwhile, the extra log log factor ensures that no $\tilde{Q}(I,\tau)$ overshoots the target. Below, we formalize the intuition.

Theorem 5.1. *On any instance I, with probability at least $1-\beta$, we have*

$$Q(I) - 4\log(GS_Q)\ln\frac{\log(GS_Q)}{\beta}\cdot\frac{\tau^*(I)}{\varepsilon} \leq \tilde{Q}(I) \leq Q(I).$$

Proof. It suffices to show that each inequality holds with probability at least $1-\frac{\beta}{2}$. For the second inequality, since $Q(I,0) \leq Q(I)$, we just need to show that $\max_j \tilde{Q}(I,\tau^{(j)}) \leq Q(I)$. By a union bound, it suffices to show that $\tilde{Q}(I,\tau) \leq Q(I)$ with probability at most $\frac{\beta}{2\log(GS_Q)}$ for each τ. This easily follows from property (2) of $Q(I,\tau)$ and the tail bound of the Laplace distribution:

$$\begin{aligned}
&\Pr[\tilde{Q}(I,\tau) > Q(I)]\\
\leq&\Pr[\tilde{Q}(I,\tau) > Q(I,\tau)]\\
=&\Pr\left[Lap\left(\log(GS_Q)\frac{\tau}{\varepsilon}\right) > \log(GS_Q)\ln\frac{\log(GS_Q)}{\beta}\cdot\frac{\tau}{\varepsilon}\right]\\
=&\frac{\beta}{2\log(GS_Q)}.
\end{aligned}$$

For the first inequality, we discuss two cases $\tau^*(I) = 0$ and $\tau^*(I) \in [2^{j-1}, 2^j]$ for some $j \geq 1$. For the first case, by property (3) of $Q(I,\tau)$, $Q(I,0) = Q(I)$. Therefore, $\tilde{Q}(I) \geq Q(I,0) = Q(I)$. Below we discuss the second case where $\tau^*(I) \in [2^{j-1}, 2^j]$. Note that $2^j \leq 2\tau^*(I)$. Let $\hat{\tau} = 2^j$. By the tail bound on the Laplace distribution, with probability at least $1-\frac{\beta}{2}$, we have

$$\begin{aligned}
\tilde{Q}(I,\hat{\tau}) \geq& Q(I,2^j) - 2\log(GS_Q)\ln\frac{\log(GS_Q)}{\beta}\cdot\frac{2^j}{\varepsilon}\\
=&Q(I) - 2\log(GS_Q)\ln\frac{\log(GS_Q)}{\beta}\cdot\frac{2^j}{\varepsilon} \qquad (9)\\
\geq&Q(I) - 4\log(GS_Q)\ln\frac{\log(GS_Q)}{\beta}\cdot\frac{\tau^*(I)}{\varepsilon}. \qquad (10)
\end{aligned}$$

Note that (9) follows the third property of $Q(I,\tau)$, and (10) is because $2^j \leq 2\tau^*(I)$. Finally, since $\tilde{Q}(I) = \max_j \tilde{Q}(I,\tau^{(j)}) \geq \tilde{Q}(I,\hat{\tau})$, the first inequality also holds with probability at least $1-\frac{\beta}{2}$. □

6 TRUNCATION FOR SJA QUERIES

In this section, we will design a $Q(I,\tau)$ with $\tau^*(I) = DS_Q(I)$ for SJA queries. Plugged into Theorem 5.1 with $\beta = 1/3$ and the definition of instance optimality, this turns R2T into an instance-optimal DP mechanism with an optimality ratio of $O(\log(GS_Q)\log\log(GS_Q)/\varepsilon)$.

For self-join-free SJA queries, each join result $q \in J(I)$ references only one tuple in R_P. Thus, the tuples in R_P are independent, i.e., removing one does not affect the sensitivities of others. This means that naive truncation (i.e., removing all $S_Q(I,t_P) > \tau$ and then summing up the rest) is a valid $Q(I,\tau)$ that satisfies the 3 properties required by R2T with $\tau^*(I) = DS_Q(I)$.

When there are self-joins, naive truncation does not satisfy property (1), as illustrated in Example 1.2, where all $S_Q(I,t_P)$'s in two neighboring instances may differ. Below we generalize the LP-based mechanism for graph pattern counting [22] to arbitrary SJA queries, and show that it satisfies the 3 properties with $\tau^*(I) = DS_Q(I)$.

Given a SJA query Q and instance I, recall that $Q(I) = \sum_{q\in J(I)}\psi(q)$, where $J(I)$ is the join results. For $k \in [|J(I)|]$, let $q_k(I)$ be the kth join result. For each $j \in [|I(R_P)|]$, let $t_j(I)$ be the jth tuple in $I(R_P)$. We use $C_j(I)$ to denote (the indices of) the set of join results that reference $t_j(I)$. More precisely,

$$C_j(I) := \{k : q_k(I) \text{ references } t_j(I)\}. \qquad (11)$$

For each $k \in [|J(I)|]$, introduce a variable u_k, which represents the weight assigned to the join result $q_k(I)$. We return the optimal solution of the following LP as $Q(I,\tau)$:

$$\begin{aligned}
\text{maximize}\quad & Q(I,\tau) = \tilde{\sum_{k\in[|J(I)|]}} u_k\\
\text{subject to}\quad & \tilde{\sum_{k\in C_j(I)}} u_k \leq \tau, j \in [|I(R_P)|],\\
& 0 \leq u_k \leq \psi(q_k(I)), k \in [|J(I)|].
\end{aligned}$$

Lemma 6.1. *For SJA queries, the $Q(I,\tau)$ defined above satisfies the 3 properties required by R2T with $\tau^*(I) = DS_Q(I)$.*

Proof. Property (2) easily follows from the constraint $u_k \leq \psi(q_k(I))$. For property (3), observe that for SJA queries, for any $j \in [|I(R_P)|]$, $S_Q(I,t_j(I)) = \sum_{k\in C_j(I)}\psi(q_k(I))$. So when $\tau \geq DS_Q(I)$, all constraints $\sum_{k\in C_j(I)} u_k \leq \tau$ are satisfied automatically and we can set $u_k = \psi(q_k(I))$ for all k.

Below, we prove property (1), i.e., for any $I \sim I'$, $Q(I,\tau)$ and $Q(I',\tau)$ differ by at most τ. W.l.o.g., assume $I \subseteq I'$. It is clear that $J(I) \subseteq J(I')$, and we order the join results in $J(I')$ in such a way that the extra join results are at the end. This means that the two LPs on I and I' share common variables $u_1,\ldots,u_{J(I)}$, while the latter has some extra variables $u_{J(I)+1},\ldots,u_{J(I')}$. Each constraint $\sum_{k\in C_j(I)} u_k \leq \tau$ in the LP on I has a counterpart $\sum_{k\in C_j(I')} u_k \leq \tau$ in the LP on I', where $C_j(I) \subseteq C_j(I')$. Let t_{j^*} be the tuple in $I'(R_P)$ that all tuples in $I'-I$ reference. Note that t_{j^*} may or may not appear in I. But in either case, the LP on I' has a constraint $\sum_{k\in C_{j^*}(I')} u_k \leq \tau$ and $C_{j^*}(I')$ contains all the extra variables in the LP on I'.

Let $\{u_k^*(I)\}_k$ be the optimal solution of the LP on I. We extend it to a solution $\{u_k(I')\}_k$ of the LP on I', by setting $u_k(I') = u_k^*(I)$ for $k \leq |J(I)|$ and $u_k(I') = 0$ for all $k > |J(I)|$. It is clear that $\{u_k(I')\}_k$ is a valid solution of the LP on I', so we have

$$Q(I',\tau) \geq \tilde{\sum_k} u_k(I') = \tilde{\sum_k} u^*(I) = Q(I,\tau).$$

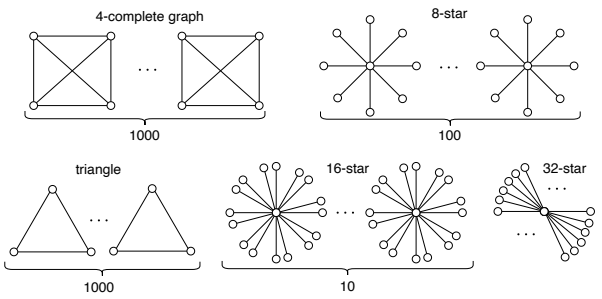

Figure 2: Example of edge counting.

For the other direction, let $\{u_k^*(\mathsf{I}')\}_k$ be an optimal solution of the LP on I'. We cut it down to a solution $\{u_k(\mathsf{I})\}_k$ of the LP on I, by setting $u_k(\mathsf{I}) = u_k^*(\mathsf{I}')$ for $k \le |J(\mathsf{I})|$ while ignoring all $u_k^*(\mathsf{I}')$ for $k > |J(\mathsf{I})|$. It is clear that $\{u_k(\mathsf{I})\}_k$ is a valid solution of the LP on I, so we have

$$Q(\mathsf{I},\tau) \ge \sum_k u_k(\mathsf{I}) \ge \sum_k u^*(\mathsf{I}') - \tau = Q(\mathsf{I}',\tau) - \tau,$$

where the second inequality follows from the observation that the constraint $\sum_{k\in C_{j^*}(\mathsf{I}')} u_k \le \tau$ in the LP on I' implies that the sum of the ignored $u_k^*(\mathsf{I}')$'s is at most τ. □

Example 6.2. We now give a step-by-step example to show how this truncation method works together with R2T. Consider the problem of edge counting under node-DP, which corresponds to the SJA query

$$Q := |\sigma_{\mathtt{ID1}<\mathtt{ID2}}(\mathtt{Node}(\mathtt{ID1}) \bowtie \mathtt{Node}(\mathtt{ID2}) \bowtie \mathtt{Edge}(\mathtt{ID1},\mathtt{ID2}))|$$

on the graph data schema introduced in Example 3.1. Note that in SQL, the query would be written as

```
SELECT count(*) FROM Node AS Node1, Node AS Node2, Edge
WHERE Edge.src = Node1.ID AND Edge.dst = Node2.ID
      AND Node1.ID < Node2.ID
```

Suppose we set $GS_Q = 2^8 = 256$. For this particular Q, this means the maximum degree of any node in any instance $\mathsf{I} \in \mathcal{I}$ is 256. We set $\beta = 0.1$ and $\varepsilon = 1$.

Now, suppose we are given an I containing 8103 nodes, which form 1000 triangles, 1000 4-cliques, 100 8-stars, 10 16-stars, and one 32-star as shown in Figure 2. The true query result is

$$Q(\mathsf{I}) = 3\times 1000 + 6\times 1000 + 8\times 100 + 16\times 10 + 32 = 9992.$$

We run R2T with $\tau^{(j)} = 2^j$ for $j = 1,\dots,8$. For each $\tau = \tau^{(j)}$, we assign a weight $u_k \in [0,1]$ to each join result (i.e., an edge) that satisfies the predicate $\mathtt{ID1} < \mathtt{ID2}$. To calculate $Q(\mathsf{I},\tau)$, we can consider the LP on each clique/star separately. For a triangle, the optimal LP solution always assigns $u_k = 1$ for each edge. For each 4-clique, it assigns 2/3 to each edge for $\tau = 2$ and 1 for $\tau \ge 4$. For each k-star, the LP optimal solution is $\min\{k,\tau\}$. Thus, the optimal LP solutions are

$$Q(\mathsf{I},2) = 1\times 3000 + \frac{2}{3}\times 6000 + 2\times 100 + 2\times 10 + 2\times 1 = 7222,$$
$$Q(\mathsf{I},4) = 1\times 3000 + 1\times 6000 + 4\times 100 + 4\times 10 + 4\times 1 = 9444,$$
$$Q(\mathsf{I},8) = 1\times 3000 + 1\times 6000 + 8\times 100 + 8\times 10 + 8\times 1 = 9888,$$
$$Q(\mathsf{I},16) = 1\times 3000 + 1\times 6000 + 8\times 100 + 16\times 10 + 16\times 1 = 9976.$$

In addition, we have $Q(\mathsf{I},0) = 0$ and $Q(\mathsf{I},\tau) = 9992$ for $\tau \ge 32$. Finally, we plug all the $Q(\mathsf{I},\tau)$'s into (7) and (8) to obtain the final output. □

7 TRUNCATION FOR SPJA QUERIES

A negative result. The correctness of the LP-based truncation method relies on a key property of SJA queries, that removing t_P will always reduce $Q(\mathsf{I})$ by $S_Q(\mathsf{I},t_P)$, which is the contribution of t_P to $Q(\mathsf{I})$. Unfortunately, the projection operator violates this property, as illustrated in the following example.

Example 7.1. Revisit the query Q in Example 1.1, where R_1 is the primary private relation, and R_2 is a secondary relation. Consider the following instance I: Set $\mathsf{I}(R_1) = \{(a_1),(a_2)\}$, $\mathsf{I}(R_2) = \{(a_i,b_j) : i\in[2], j\in[m]\}$. Then $S_Q(\mathsf{I},(a_1)) = S_Q(\mathsf{I},(a_2)) = m$, $Q(\mathsf{I}) = 2m$, and $DS_Q(\mathsf{I}) = m$.

Now, we add a projection operator, changing the query to

$$Q' := |\pi_{x_2}(R_1(x_1) \bowtie R_2(x_1,x_2))|.$$

Both (a_1) and (a_2) contribute m to $Q(\mathsf{I})$ but their contributions "overlap", thus removing either will not affect the query result, i.e., $DS_{Q'}(\mathsf{I}) = 0$. □

Intuitively, a projection reduces the query answer, hence its sensitivity, so it requires less noise. However, it makes achieving instance optimality harder because the optimality target, $DS_Q(\mathsf{I})$, may get a lot smaller, as illustrated in the example above. In particular, the second equality in (6) no longer holds (the first equality is the definition of $DS_Q(\mathsf{I})$), and $DS_Q(\mathsf{I})$ may be smaller than any $S_Q(\mathsf{I},t_P)$. We formalize this intuition with the following negative result:

Theorem 7.2. *Let Q' be the query in Example 7.1. For any GS_Q, there is a set of instances $\mathcal{I}$ with global sensitivity GS_Q such that, for any functions $M, f : \mathcal{I} \to \mathbb{R}$, if $\Pr[|M(\mathsf{I}) - Q'(\mathsf{I})| \le f(\mathsf{I})\cdot DS_{Q'}(\mathsf{I})] \ge 2/3$, then M is not ε-DP for any $\varepsilon > \frac{1}{2}\ln(GS_Q/2)$.*

Proof. We build the set of instances $\mathcal{I}$ as follows. First, put the empty instance I_0 into $\mathcal{I}$. Then, for any $m \in [GS_Q]$, construct an I_m with $\mathsf{I}_m(R_1) = \{(a_1),(a_2)\}$, $\mathsf{I}_m(R_2) = \{(a_i,b_j) : i\in[2], j\in[m]\}$. Note that $Q'(\mathsf{I}_m) = m$, and $DS_{Q'}(\mathsf{I}_m) = 0$ since removing either (a_1) or (a_2) will not affect the query result. Finally, for each I_m, remove (a_1) (and all referencing tuples) and add the resulting instance to $\mathcal{I}$. It can be verified that the global sensitivity of $\mathcal{I}$ is GS_Q. Meanwhile, for any $m \in [GS_Q]$, I_m and I_0 are 2-hop neighbors, so if M is ε-DP, then

$$\Pr[M(\mathsf{I}_m) = y] \le e^{2\varepsilon}\Pr[M(\mathsf{I}_0) = y],$$

for any y, by the *group privacy* property of DP [15].

The instance-optimality guarantee implies that for every $m \in [GS_Q]$,

$$\Pr[M(\mathsf{I}_m) = m] \ge 2/3.$$

Consider I_0. On the one hand,

$$\Pr[M(\mathsf{I}_0) \ne 0] \le 1/3. \tag{12}$$

On the other hand,

$$\begin{aligned}\Pr[M(\mathsf{I}_0) \neq 0] &\geq \Pr[M(\mathsf{I}_0) = 1] + \cdots + \Pr[M(\mathsf{I}_0) = GS_Q] \\ &\geq \sum_{m=1}^{GS_Q} e^{-2\varepsilon} \Pr[M(\mathsf{I}_m) = m] \\ &\geq \sum_{m=1}^{GS_Q} e^{-2\varepsilon} \cdot \frac{2}{3} = \frac{2GS_Q}{3e^{2\varepsilon}},\end{aligned}$$

which contradicts (12) when $\varepsilon \geq \frac{1}{2}\ln(GS_Q/2)$. □

Indirect sensitivity. Recall the definition of $S_Q(\mathsf{I}, t_P)$ as in (4). However, for an SPJA query, we have $Q(\mathsf{I}) = \prod_{q \in \pi_{\mathsf{y}} J(\mathsf{I})} \psi(q)$ instead of $Q(\mathsf{I}) = \prod_{q \in J(\mathsf{I})} \psi(q)$ thus (6) no longer holds. This means that, while $S_Q(\mathsf{I}, t_P)$ is still the contribution of t_P to $Q(\mathsf{I})$, it is "indirect": The overlapping contributions should be counted only once due to the projection operator removing duplicates.

We now define the *indirect sensitivity* for an instance I:

$$IS_Q(\mathsf{I}) = \max_{t_P \in \mathsf{I}(R_P)} S_Q(\mathsf{I}, t_P).$$

It should be clear that $IS_Q(\mathsf{I}) \geq DS_Q(\mathsf{I})$ due to the overlapping; in the extreme case shown in Example 7.1, we have $IS_Q(\mathsf{I}) = m$ but $DS_Q(\mathsf{I}) = 0$. Below we give a truncation method for SPJA queries with $\tau^*(\mathsf{I}) = IS_Q(\mathsf{I})$. When plugged into R2T, this yields a DP mechanism with error $O(\log(GS_Q)\log\log(GS_Q)IS_Q(\mathsf{I})/\varepsilon)$. This is not instance-optimal, which is unachievable by Theorem 7.2 anyway. Note that for SJA queries, we have $\mathsf{y} = var(J)$, and $DS_Q(\mathsf{I}) = IS_Q(\mathsf{I})$ in this case.

Truncation method. We modify the LP-based truncation method from Section 6 to handle SPJA queries. Let $p_l(\mathsf{I})$ be the l-th result in $\pi_{\mathsf{y}} J(\mathsf{I})$, $q_k(\mathsf{I})$ the k-th result in $J(\mathsf{I})$. To formalize the relationship of the query results before and after the projection, we use $D_l(\mathsf{I})$ to denote (the indices of) the join results corresponding to the projected result $p_l(\mathsf{I})$, i.e.,

$$D_l(\mathsf{I}) := \{j : p_l = \pi_{\mathsf{y}} q_j(\mathsf{I})\},$$

while $C_j(\mathsf{I})$ is still defined as in (11). Then $S_Q(\mathsf{I}, t_j)$ can be rewritten as

$$S_Q(\mathsf{I}, t_j) = \sum_{k \in C_j(\mathsf{I})} \psi(q_k(\mathsf{I})).$$

Now, we define a new LP. For each $l \in [|\pi_{\mathsf{y}} J(\mathsf{I})|]$, we introduce a new variable $v_l \in [0, \psi(p_l(\mathsf{I}))]$, which represents the weight assigned to the projected result $p_l(\mathsf{I})$. For each $k \in [|J(\mathsf{I})|]$, we still use a variable $u_k(\mathsf{I}) \in [0, \psi(q_k(\mathsf{I}))]$ to represent the weight assigned to $q_k(\mathsf{I})$. We keep the same truncation constraints on the u_k's, while adding the constraint that a the weight of a projected result should not exceed the total weights of all its corresponding join results. Then we try to maximize the projected results. More precisely, the new LP is

$$\begin{aligned}\text{maximize} \quad & Q(\mathsf{I}, \tau) = \sum_{l \in [|\pi_{\mathsf{y}} J(\mathsf{I})|]} v_l \\ \text{subject to} \quad & v_l \leq \sum_{k \in D_l(\mathsf{I})} u_k \\ & \sum_{k \in C_j(\mathsf{I})} u_k \leq \tau, j \in [|\mathsf{I}(R_P)|], \\ & 0 \leq u_k \leq \psi(q_k(\mathsf{I})), k \in [|J(\mathsf{I})|] \\ & 0 \leq v_l \leq \psi(p_l(\mathsf{I})), l \in [|\pi_{\mathsf{y}} J(\mathsf{I})|].\end{aligned}$$

We can show that this modified LP yields a valid truncation method for SPJA queries:

Lemma 7.3. *For SPJA queries, the $Q(\mathsf{I}, \tau)$ defined above satisfies the 3 properties required by R2T with $\tau^*(\mathsf{I}) = IS_Q(\mathsf{I})$.*

Proof. First, same as SJA queries, property (2) holds due to the constraint $v_l \leq \psi(p_l(\mathsf{I}))$. For property (3), we have $S_Q(\mathsf{I}, t_j) = \prod_{k \in C_j(\mathsf{I})} \psi(q_k(\mathsf{I}))$. Then with same argument as in the proof of Lemma 6.1, we can show that the property holds with $\tau^*(\mathsf{I}) = IS_Q(\mathsf{I})$. Finally consider property (1). For any $\mathsf{I} \sim \mathsf{I}'$, $\mathsf{I} \subseteq \mathsf{I}'$, it is easy to see that $J(\mathsf{I}) \subseteq J(\mathsf{I}')$ and all different projection results are in C_{j^*} for some $j^* \in [|\mathsf{I}(R_P)|]$. Then the same line of reasoning as in the proof of Lemma 6.1 proves property (1). □

8 MULTIPLE PRIMARY PRIVATE RELATIONS

Now we consider the case with $k \geq 2$ primary private relations $R_P^1, \ldots, R_P^k$. In this case, two instances are considered neighbors if one can be obtained from the other by deleting a set of tuples, all of which reference the same tuple that belongs to some $R_P^i, i \in [k]$. We reduce it to the case with only one primary private relation as follows. Add a new column `ID` to every $\mathsf{I}(R_P^i), i \in [k]$, and assign unique identifiers to all tuples in these relations. Next, we construct a new relation $R_P(\mathtt{ID})$, whose physical instance $\mathsf{I}(R_P)$ consists of all these identifiers. For each R_P^i, we add an FK constraint from its `ID` column to reference the `ID` column of R_P. Note that this FK reference relationship is actually a bijection between the `ID` column in R_P and all the identifiers in the primary private relations. Now, we designate R_P as the only primary private relation, while $R_P^i, i \in [k]$ all become secondary private relations. The original secondary private relations, i.e., those having FK references to the R_P^i's directly or indirectly, are still secondary private relations.

It is not hard to see that (1) the query answer is not affected by this schema change; (2) two instances in the original schema are neighbors if and only if they are neighbors in the new schema; and (3) the join results that reference any tuple $t \in \mathsf{I}(R_P^i), i \in [k]$ are the same as those that reference $t_P \in \mathsf{I}(R_P)$, where t_P and t have the same identifier. Thus, both the privacy and utility guarantees of our algorithm continue to hold.

Finally, it is worth pointing out that the reduction above is conceptual; in the actual implementation, there is no need to construct the new primary private relation and the additional `ID` columns, as illustrated in Example 9.1 of the next section.

9 SYSTEM IMPLEMENTATION

Based on the R2T algorithm, we have implemented a system on top of PostgreSQL and CPLEX. The system structure is shown in Figure 3. The input to our system is any SPJA query written in SQL, together with a designated primary private relation R_P (interestingly, while R2T satisfies the DP policy with FK constraints, the algorithm itself does not need to know the PK-FK constraints).

The system supports `SUM` and `COUNT` aggregation. Our SQL parser first unpacks the aggregation into a reporting query so as to find $\psi(q_k(\mathsf{I}))$ for each join result, as well as $C_j(\mathsf{I})$, which stores the referencing relationships between tuples in $\mathsf{I}(R_P)$ and $J(\mathsf{I})$.

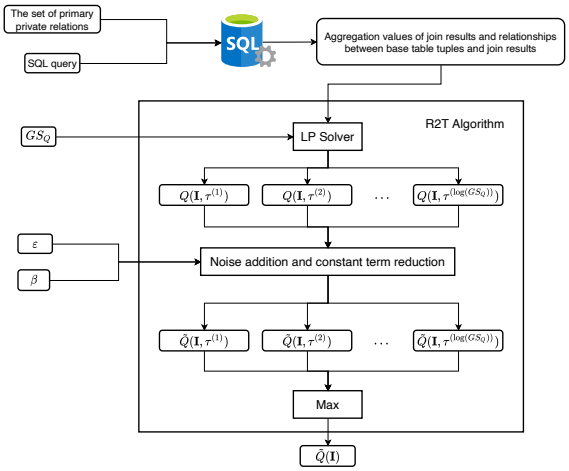

Figure 3: System structure.

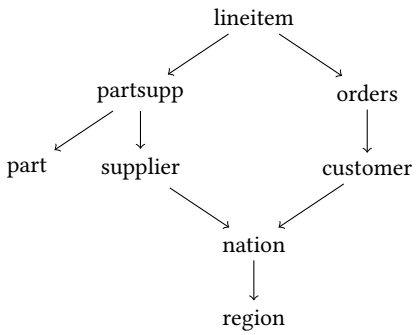

Figure 4: The foreign-key graph of TPC-H schema.

Example 9.1. Suppose we use the TPC-H schema (shown in Figure 4), where we designate `Supplier` and `Customer` as primary private relations. Consider the following query:

```
SELECT SUM(price * (1 - discount))
FROM Supplier, Lineitem, Orders, Customer
WHERE Supplier.SK = Lineitem.SK AND Lineitem.OK = Orders.OK
  AND Orders.CK = Customer.CK
  AND Orders.orderdate >= '2020-08-01'
```

We rewrite it as

```
SELECT Supplier.SK, Customer.CK, price * (1 - discount)
FROM Supplier, Lineitem, Orders, Customer
WHERE Supplier.SK = Lineitem.SK AND Lineitem.OK = Orders.OK
  AND Orders.CK = Customer.CK
  AND Orders.orderdate >= '2020-08-01'
```

The `price * (1 - discount)` column in the query results gives all the $\psi(q_k(\mathbf{I}))$ values, while `Supplier.SK` and `Customer.CK` yield the referencing relationships from each supplier and customer to all the join results they contribute to. □

We execute the rewritten query in PostgreSQL, and export the query results to a file. Then, an external program is invoked to construct the $\log(GS_Q)$ LPs from the query results, which are then solved by CPLEX. Finally, we use R2T to compute a privatized output.

The computation bottleneck is the $\log(GS_Q)$ LPs, each of which contains $|J(\mathbf{I})|$ variables and $|J(\mathbf{I})| + |\mathbf{I}(R_P)|$ constraints. This takes polynomial time, but can still be very expensive in practice. One immediate optimization is to solve them in parallel. Below we present another effective technique to speed up the process.

Algorithm 1: R2T with early stop

Input: $\mathbf{I}$, Q,R_P, GS_Q

1 $\tilde{Q}(\mathbf{I}) \leftarrow 0$;
2 **for** $\tau^{(j)} \leftarrow GS_Q, GS_Q/2, \ldots, 1$ **do in parallel**
3 $\quad T^{(j)} \leftarrow Lap\left(\log(GS_Q)\frac{\tau^{(j)}}{\varepsilon}\right) - \log(GS_Q)\ln\frac{\log(GS_Q)}{\beta}\cdot\frac{\tau^{(j)}}{\varepsilon}$;
4 $\quad$ **for** $t \leftarrow 1, 2, \ldots$ **do**
5 $\quad\quad$ **if** $\hat{Q}^{(t)}(\mathbf{I},\tau^{(j)})$ *achieves the optimal* **then**
6 $\quad\quad\quad \tilde{Q}(\mathbf{I}) \leftarrow \max(\tilde{Q}(\mathbf{I}), \hat{Q}^{(t)}(\mathbf{I},\tau^{(j)}) + T^{(j)})$;
7 $\quad\quad\quad$ Break;
8 $\quad\quad$ **else if** $\hat{Q}^{(t)}(\mathbf{I},\tau^{(j)}) + T^{(j)} \le \tilde{Q}(\mathbf{I})$ **then**
9 $\quad\quad\quad$ Break;
10 $\quad\quad$ **end**
11 $\quad$ **end**
12 **end**
13 **return** $\tilde{Q}(\mathbf{I})$;

Early stop. The key observation is that R2T returns the maximum of $O(\log(GS_Q))$ maximization LPs (masked by some noise and reduced by a factor), and most LP solvers (e.g., CPLEX) for maximization problems use some iterative search technique to gradually approach the optimum from below, namely, these $O(\log(GS_Q))$ LP solvers all "race to the top". Thus, we will not know the winner until they all stop.

To cut down the unnecessary search, the idea is to flip the problem around. Instead of solving the primal LPs, we solve their duals. By LP duality, the dual LP has the same optimal solution as the primal, but importantly, the LP solver will approach the optimal solution from above, namely, we have a gradually decreasing upper bound for the optimal solution of each LP. This allows us to terminate those LPs that have no hope to be the winner. The optimized R2T algorithm, shown in Algorithm 1, also uses the trick that the noises are generated before we start running the LP solvers, so that we know when to terminate.

In Algorithm 1, we use t to denote the iterations of the LP solver, and use $\hat{Q}^{(t)}(\mathbf{I},\tau)$ to denote the solution to the dual LP in the t-th iteration. A technicality is that in line 1, we should initialize $\tilde{Q}(\mathbf{I})$ to $Q(\mathbf{I},0)$ to be consistent with the R2T algorithm, but $Q(\mathbf{I},0) = 0$ for all the truncation methods described in this paper.

When there are not enough CPU cores to solve all LPs in parallel, we choose to start with those with a larger τ in line 3 of Algorithm 1. This is based on our observation that those LPs tend to terminate

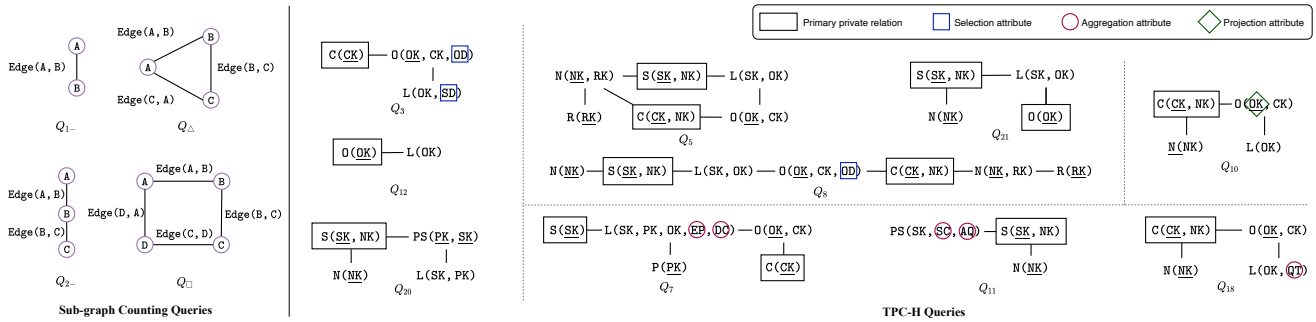

Figure 5: The structure of queries.

faster. This is very intuitive: when τ is larger, the optimal solution is also higher, thus the LP solver for the dual can terminate earlier.

Dataset	Deezer	Amazon1	Amazon2	RoadnetPA	RoadnetCA
Nodes	144,000	262,000	335,000	1,090,000	1,970,000
Edges	847,000	900,000	926,000	1,540,000	2,770,000
Maximum degree	420	420	549	9	12
Degree upper bound D	1,024	1,024	1,024	16	16

Table 1: Graph datasets used in the experiments.

10 EXPERIMENTS

We conducted experiments on two types of queries: graph pattern counting queries under node-DP and general SPJA queries with FK constraints, with the former being an important special case of the latter. For graph pattern counting queries, we compare R2T with naive truncation with smooth sensitivity (NT) [22], smooth distance estimator (SDE) [8], recursive mechanism (RM) [9], and the LP-based mechanism (LP) [22]. For general SPJA queries, we compare with the local sensitivity-based mechanism (LS) [37].

10.1 Setup

Queries. For graph pattern counting queries, we used four queries: edge counting Q_{1-}, length-2 path counting Q_{2-}, triangle counting $Q_{\triangle}$, and rectangle counting $Q_{\square}$. For SPJA queries, we used 10 queries from the TPC-H benchmark, whose structures are shown in Figure 5. These queries involve a good mix of selection, projection, join, and aggregation. We removed all the group-by clauses from the queries — a brief discussion on this is provided at the end of the paper.

Datasets. For graph pattern counting queries, we used 5 real world networks datasets: Deezer, Amazon1, Amazon2, RoadnetPA and RoadnetCA. Deezer collects the friendships of users from the music streaming service Deezer. Amazon1 and Amazon2 are two Amazon co-purchasing networks. RoadnetPA and RoadnetCA are road networks of Pennsylvania and California, respectively. All these datasets are obtained from SNAP [24]. Table 1 shows the basic statistics of these datasets.

Most algorithms need to assume a GS_Q in advance. Note that the value of GS_Q should not depend on the instance, but may use some background knowledge for a particular class of instances. Thus, for the three social networks, we set a degree upper bound of $D = 1024$, while for the two road networks, we set $D = 16$. Then we set GS_Q as the maximum number of graph patterns containing any node. This means that $GS_{Q_{1-}} = D$, $GS_{Q_{2-}} = GS_{Q_{\triangle}} = D^2$, and $GS_{Q_{\square}} = D^3$. For TPC-H queries, we used datasets of scale $2^{-3}, 2^{-2}, \ldots, 2^3$. The one with scale 1 (default scale) has about 7.5 million tuples, and we set $GS_Q = 10^6$.

The LP mechanism requires a truncation threshold τ, but [22] does not discuss how this should be set. Initially, we used a random threshold uniformly chosen from $[1, GS_Q]$. This turned out to be very bad as with constant probability, the picked threshold is $\Omega(GS_Q)$, which makes these mechanisms as bad as the naive mechanism that adds GS_Q noise. To achieve better results, as in R2T, we consider $\{2, 4, 8, \ldots, GS_Q\}$ as the possible choices. Similarly, NT and SDE need a truncation threshold θ on the degree, and we choose one from $\{2, 4, 8, \ldots, D\}$ randomly.

Experimental environment. All experiments were conducted on a Linux server with a 24-core 2.2GHz Intel Xeon CPU and 256GB of memory. Each program was allowed to use at most 10 threads and we set a time limit of 6 hours for each run. Each experiment was repeated 100 times and we report the average running time. The errors are less stable due to the random noise, so we remove the best 20 and worst 20 runs, and report the average error of the remaining 60 runs. The failure probability β in R2T is set to 0.1. The default DP parameter is $\varepsilon = 0.8$.

10.2 Graph Pattern Counting Queries

Utility and efficiency. The errors and running times of all mechanisms over the graph pattern counting queries are shown in Table 2. These results indicate a clear superiority of R2T in terms of utility, offering order-of-magnitude improvements over other methods in many cases. What is more desirable is its robustness: In all the 20 query-dataset combinations, R2T consistently achieves an error below 20%, while the error is below 10% in all but 3 cases. We also notice that, given a query, R2T performs better in road networks than social networks. This is because the error of R2T is proportional to $DS_Q(\mathbf{I})$ by our theoretical analysis. Thus the relative error is proportional to $DS_Q(\mathbf{I})/|Q(\mathbf{I})|$. Therefore, larger and sparser graphs, such as road networks, lead to smaller relative errors.

In terms of running time, all mechanisms are reasonable, except for RM and SDE. RM can only complete within the 6-hour time

Dataset		Deezer		Amazon1		Amazon2		Roadnet – PA		Roadnet – CA	
Result type		Relative error(%)	Time(s)	Relative error(%)	Time(s)	Relative error(%)	Time(s)	Relative error(%)	Time(s)	Relative error(%)	Time(s)
q_{1-}	Query result	847,000	1.28	900,000	1.52	926,000	1.62	1,540,000	1.51	2,770,000	2.64
	R2T	0.535	12.3	0.557	15.6	0.432	16.2	0.0114	26.8	0.00635	48.7
	NT	59.1	18.1	101	29.3	125	40.4	1,370	21.9	1,410	39.7
	SDE	548	9,870	363	4,570	286	1,130	55.2	105	81.8	292
	LP	14.3	16.9	5.72	14.7	6.75	14.4	3.6	28.3	3.02	54
q_{2-}	Query result	21,800,000	13.8	9,120,000	11.8	9,750,000	13.8	3,390,000	6.39	6,000,000	6.06
	R2T	6.64	356	12.2	170	9.06	196	0.0539	80.2	0.0352	145
	NT	116	21.0	398	28.4	390	41.0	6,160	23.2	6,530	44.2
	SDE	8,900	9,870	5,110	4,570	1,930	1,130	211	104	228	296
	LP	35.9	8,820	23.2	3,600	27.8	461	11.1	148	13.3	404
$q_{\triangle}$	Query result	794,000	4.53	718,000	5.03	667,000	4.20	67,200	2.96	121,000	5.17
	R2T	5.58	17.3	1.27	18.8	2.03	19.9	0.102	4.21	0.061	7.5
	NT	782	23.0	1,660	31.7	1,920	41.0	110,000	23.3	105,000	45.0
	SDE	67,300	9,880	26,000	4,570	9,600	1,130	4,150	106	3,830	297
	LP	24.6	131	12.8	18.2	14.2	18.3	0.104	3.95	0.0625	7.06
	RM	Over time limit						0.0388	1,280	0.0193	2,550
$q_{\square}$	Query result	11,900,000	74.3	2,480,000	21.6	3,130,000	15.6	158,000	4.50	262,000	10.1
	R2T	16.9	289	6.29	70.5	10.5	86.8	0.0729	8.18	0.0638	16.2
	NT	3,750	57.6	30,700	35.8	26,100	50.6	319,000	24.8	368,000	45.0
	SDE	6,970,000	9,930	11,400,000	4,580	202,000	1,140	10,300	108	9,130	300
	LP	92.6	2,530	94.8	70.4	77.8	81.2	0.223	7.83	0.165	14.2
	RM	Over time limit						0.0217	10,500	Over time limit	

Table 2: Comparison between R2T, naive truncation with smooth sensitivity (NT), smooth distance estimator (SDE), LP-based Mechanism (LP), and recursive mechanism (RM) on graph pattern counting queries.

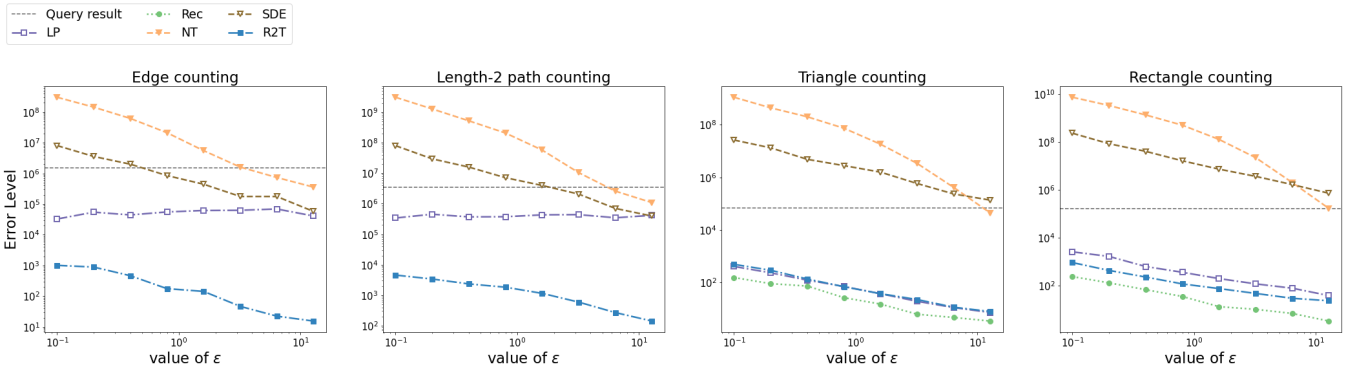

Figure 6: Error levels of various mechanisms on graph pattern counting queries various values of ε.

Query		Q_{1-}	Q_{2-}	$Q_{\triangle}$	$Q_{\square}$
Query result		926,000	9,750,000	667,000	3,130,000
R2T		4,000	883,000	13,500	328,000
LP	$\tau = GS_Q$	1,440	1,580,000	1,290,000	1,370,000,000
	$\tau = GS_Q/8$	2,100	181,000	157,000	140,000,000
	$\tau = GS_Q/64$	110,000	259,000	15,100	25,800,000
	$\tau = GS_Q/512$	645,000	1,260,000	2,790	2,630,000
	$\tau = GS_Q/4096$	810,000	3,950,000	2,090	274,000
	$\tau = GS_Q/32768$	911,000	7,580,000	92,300	48,700
	$\tau = GS_Q/262144$	924,000	9,340,000	459,000	76,400
	Average error	62,500	2,710,000	94,900	2,430,000

Table 3: Error levels of R2T and LP-based mechanism (LP) with different τ.

limit on 3 cases, although it achieves very small errors on these 3 cases. SDE is faster than RM but runs a bit slower than others. It is also interesting to see that R2T sometimes even runs faster than

Dataset	Deezer	Amazon1	Amazon2	RoadnetPA	RoadnetCA
w early stop	289	70.5	86.8	8.18	16.2
w/o early stop	28,700	537	422	12.8	16.4
Speed up	99.3×	7.62 ×	4.86×	1.56×	1.01×

Table 4: Running times of R2T with and without early stop.

LP, despite the fact that R2T needs to solve $O(\log GS_Q)$ LPs. This is due to the early stop optimization: The running time of R2T is determined by the LP that corresponds to the near-optimal τ, which often happens to be one of the LPs that can be solved fastest.

Privacy parameter ε. Next, we conducted experiments to see how the privacy parameter ε affects various mechanisms. We tested different queries on Roadnet – PA where we vary ε from 0.1 to 12.8. We plot the results in Figure 6, where we also plot the query result to help see the utilities of the mechanisms. The first message from the plot is the same as before, that both R2T and RM achieve high

Query type		Single primary private relation			Multiple primary private relations			Aggregation			Projection
Query		Q_3	Q_{12}	Q_{20}	Q_5	Q_8	Q_{21}	Q_7	Q_{11}	Q_{18}	Q_{10}
Query result	Value	2,890,000	6,000,000	6,000,000	240,000	1,830,000	6,000,000	218,000,000	2,000,000	153,000,000	1,500,000
	Time(s)	1.6	1.24	1.25	2.51	1.41	2.32	3.22	0.29	2.21	0.32
R2T	Relative error(%)	0.254	0.0229	0.579	1.626	1.92	0.654	0.607	1.82	0.132	0.174
	Time(s)	18.9	28.2	24.5	8.42	39.6	124	140	4.41	42.7	8.77
LS	Relative error(%)	38.8	16.3	15.4	Not supported						
	Time(s)	19.2	25.8	24.4							

Table 5: Comparison between R2T and local-sensitivity based mechanism (LS) on SQL queries.

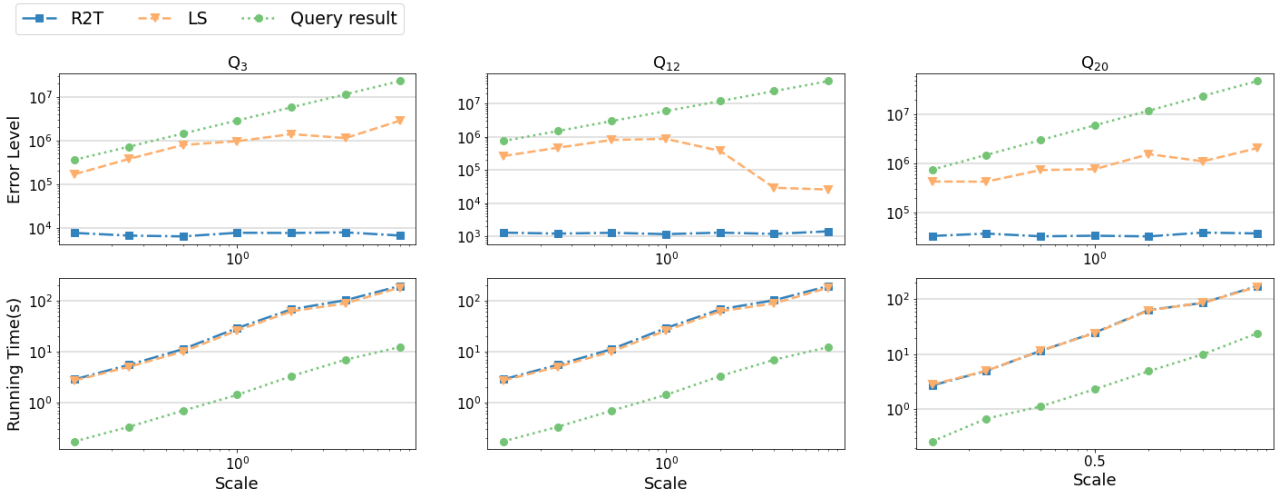

Figure 7: Running times and error levels of R2T and local-sensitivity based mechanism (LS) for different data scales.

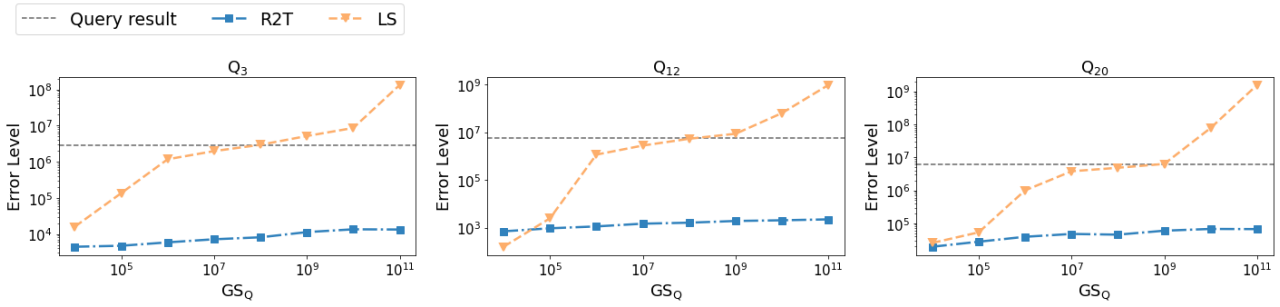

Figure 8: Error levels of R2T and local-sensitivity based mechanism (LS) with different GS_Q.

utility (but RM spends 300x more time). NT and SDE lose utility (i.e., error larger than query result) except for very large ε. LP achieves similar utility as R2T on $Q_\triangle$ and $Q_\square$, but is much worse on Q_{1-} and Q_{2-}. In particular, a higher ε does not help LP on these two queries, because the bias (further controlled by a randomly selected τ) dominates the error for these two queries.

Selection of τ. In the next set of experiments, we dive deeper and see how sensitive the utility is with respect to the truncation threshold τ. We tested the queries on Amazon2 and measured the error of the LP-based mechanism [22] with different τ. For each query, we tried various τ from 2 to GS_Q and compare their errors with R2T. The results are shown in Table 3, where the optimal error is marked in gray. The results indicate that the error is highly sensitive to τ, and more importantly, the optimal choice of τ closely depends on the query, and there is no fixed τ that works for all cases. On the other hand, the error of R2T is within a small constant factor (around 6) to the optimal choice of τ, which is exactly the value of instance-optimality.

Early stop optimization. We also did some experiments to compare the running time of R2T with and without the early stop optimization. Here, we ran $Q_\square$ over different datasets and the results are shown in Table 4. From this table, we can see the early stop is

particularly useful in cases with long running times. In these cases, one or two LPs, which do not correspond to the optimal choice of τ, take a long time to run, and early stop is able to terminate these LPs as soon as possible.

10.3 SPJA Queries

Utility and efficiency. We tested 10 queries from the TPC-H benchmark comparing R2T and LS, and the results are shown in Table 5. We see that R2T achieves order-of-magnitude improvements over LS in terms of utility, with similar running times. More importantly, R2T supports a variety of SPJA queries that are not supported by LS, with robust performance across the board.

Scalability. To examine the effects as the data scale changes, we used TPC-H datasets with scale factors ranging from 2^{-3} to 2^3 with Q_3, Q_{12} and Q_{20}. We compare both the errors and running times of R2T and LS. The results are shown in Figure 7. From the results, we see that the error of R2T barely increases with the data size. The reason is that our error only depends on $DS_Q(\mathbf{I})$, which does not change much by the scale of TPC-H data. On the other hand, the behavior of LS is more complicated. For Q_3 and Q_{20}, its error increases with the data size; for Q_{12}, its error increases first but then decreases later. This is because LS runs an SVT on the sensitivities of tuples to choose τ, which is closely related to the distribution of tuples' sensitivities. This is another indication that selecting a near-optimal τ is not an easy task. In terms of running time, both mechanisms have the running time linearly increase with the data size, which is expected.

Dependency on GS_Q. Our last set of experiments examine the effect GS_Q brings to the utilities of R2T and LS. We conducted experiments using Q_3, Q_{12}, Q_{20} with different values of GS_Q. The results are shown in Figure 8. When GS_Q is small, the errors of these two mechanisms are very close. When GS_Q increases, the error of LS increases rapidly, and loses the utility (error larger than query result) very soon. Meanwhile, the error of R2T increases very slowly with GS_Q. This confirms our analysis that the error of LS grows near linearly as GS_Q, while that of R2T grows logarithmically. The important consequence is that, with R2T, one can be very conservative in setting the value of GS_Q. This gives the DBA a much easier job, in case s/he has little idea on what datasets the database is expected to receive. Meanwhile, recall that GS_Q is public information, so using a larger GS_Q reveals less information about the private dataset.

11 FUTURE WORK

One interesting future direction is how to handle group-by queries. One simple solution is to just convert a group-by query (e.g., group-by `NATION`) into multiple queries, each with a different predicate (e.g, `NATION =' US'`). Note that the privacy budget will also have to be split to the groups according to various DP composition theorems [15]. However, there is potential to do better by computing the answers for all groups in one shot. For self-join-free queries, this is precisely the mean estimation problem in high dimensions, which has received much attention in the statistics and machine learning community [2, 3, 18, 27, 33]. Self-joins add another challenge that is definitely worth studying in more depth, which may expose a deeper connection between statistics/machine learning and query processing.

ACKNOWLEDGMENTS

This work has been supported by HKRGC under grants 16201318, 16201819, and 16205420; by National Science Foundation under grants 2016393; and by DARPA and SPAWAR under contract N66001-15-C-4067. We would also like to thank Bin Wu for his technical support and the anonymous reviewers who have made valuable suggestions on improving the presentation of the paper.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] Kareem Amin, Alex Kulesza, Andres Munoz, and Sergei Vassilvtiskii. Bounding user contributions: A bias-variance trade-off in differential privacy. In *International Conference on Machine Learning*, pages 263–271. PMLR, 2019.

[3] Galen Andrew, Om Thakkar, H Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. *arXiv preprint arXiv:1905.03871*, 2019.

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

[5] Hilal Asi and John C Duchi. Instance-optimality in differential privacy via approximate inverse sensitivity mechanisms. *Advances in Neural Information Processing Systems*, 33, 2020.

[6] Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Satyen Kale, Frank McSherry, and Kunal Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 273–282, 2007.

[7] Jaroslaw Błasiok, Mark Bun, Aleksandar Nikolov, and Thomas Steinke. Towards instance-optimal private query release. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2480–2497. SIAM, 2019.

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

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

[10] Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 123–138, 2016.

[11] Apple Differential Privacy Team. Learning with privacy at scale. https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf. December 2017.

[12] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *NeurIPS*, 2017.

[13] Wei Dong and Ke Yi. Residual sensitivity for differentially private multi-way joins. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2021.

[14] Wei Dong and Ke Yi. A nearly instance-optimal differentially private mechanism for conjunctive queries. In *Proc. ACM Symposium on Principles of Database Systems*, 2022.

[15] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[16] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[17] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems*, pages 2339–2347, 2012.

[18] Ziyue Huang, Yuting Liang, and Ke Yi. Instance-optimal mean estimation under differential privacy. In *NeurIPS*, 2021.

[19] Noah Johnson, Joseph P Near, and Dawn Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[20] Vishesh Karwa and Salil Vadhan. Finite sample differentially private confidence intervals. In *9th Innovations in Theoretical Computer Science Conference (ITCS*

2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
[21] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11): 1146–1157, 2011.
[22] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography Conference*, pages 457–476. Springer, 2013.
[23] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment*, 12(11):1371–1384, 2019.
[24] Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection (2014). *URL http://snap. stanford. edu/data*, page 49, 2016.
[25] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB journal*, 24(6):757–781, 2015.
[26] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *2008 IEEE 24th international conference on data engineering*, pages 277–286. IEEE, 2008.
[27] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.
[28] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009.
[29] Arjun Narayan and Andreas Haeberlen. Djoin: Differentially private join queries over distributed databases. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 149–162, 2012.
[30] Aleksandar Nikolov, Kunal Talwar, and Li Zhang. The geometry of differential privacy: the sparse and approximate cases. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 351–360, 2013.
[31] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84, 2007.
[32] Catuscia Palamidessi and Marco Stronati. Differential privacy for relational algebra: Improving the sensitivity bounds via constraint systems. In *QAPL*, 2012.
[33] Venkatadheeraj Pichapati, Ananda Theertha Suresh, Felix X Yu, Sashank J Reddi, and Sanjiv Kumar. Adaclip: Adaptive clipping for private sgd. *arXiv preprint arXiv:1908.07643*, 2019.
[34] Davide Proserpio, Sharon Goldberg, and Frank McSherry. Calibrating data to sensitivity in private data analysis. *Proceedings of the VLDB Endowment*, 7(8), 2014.
[35] Wahbeh Qardaji, Weining Yang, and Ninghui Li. Practical differentially private release of marginal contingency tables. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1435–1446.
[36] Wahbeh Qardaji, Weining Yang, and Ninghui Li. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment*, 6(14):1954–1965, 2013.
[37] Yuchao Tao, Xi He, Ashwin Machanavajjhala, and Sudeepa Roy. Computing local sensitivities of counting queries with joins. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 479–494, 2020.
[38] Salil Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017.
[39] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private sql with bounded user contribution. *Proceedings on privacy enhancing technologies*, 2020(2):230–250, 2020.
[40] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on knowledge and data engineering*, 23(8): 1200–1214, 2010.
[41] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Private release of graph statistics using ladder functions. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 731–745, 2015.
[42] Xiaojian Zhang, Rui Chen, Jianliang Xu, Xiaofeng Meng, and Yingtao Xie. Towards accurate histogram publication under differential privacy. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 587–595. SIAM, 2014.

A THE ERROR BOUND OF [37]

[37] makes truncation by tuples' sensitivity and the algorithm to find the truncation threshold is based on an upper bound on tuple sensitivities. It is denoted as ℓ in [37], but we observe that this is just the global sensitivity. So we denote this given upper bound as GS_Q. Note that a trivial method is to set $\tau = GS_Q$, which has no bias while the error is $O(GS_Q \log(1/\beta)/\varepsilon)$ with probability $1-\beta$. The mechanism for choosing τ in [37] is DP, but we show below that it is not much better than this naive choice, on *any* instance I. More precisely, we show that its error is $\Omega(GS_Q/(\log(GS_Q)\varepsilon))$ with at least constant probability.

Recall, the algorithm first constructs a DP-version of query result

$$\hat{Q}(\mathsf{I}) = Q(\mathsf{I}) + Lap(\frac{GS_Q}{\varepsilon}).$$

Based on this, we can see,

$$\Pr[\hat{Q}(\mathsf{I}) \geq Q(\mathsf{I}) + GS_Q/\varepsilon] = \frac{1}{2e}.$$

Then, recall $Q(\mathsf{I},\tau)$ is the query result after truncating the tuples with sensitivity larger than τ, and $Q(\mathsf{I},\tau) \leq Q(\mathsf{I})$. Then, when $\hat{Q}(\mathsf{I}) \geq Q(\mathsf{I}) + GS_Q/\varepsilon$, for any $\tau \ddot{Y} GS_Q/(6\log(GS_Q/\beta))$,

$$\begin{aligned}
&\Pr[Q(\mathsf{I},\tau) + Lap(2\tau/\varepsilon) + Lap(4\tau/\varepsilon) \geq \hat{Q}(\mathsf{I})] \\
\leq &\Pr[Q(\mathsf{I}) + Lap(2\tau/\varepsilon) + Lap(4\tau/\varepsilon) \geq \hat{Q}(\mathsf{I})] \\
\leq &\Pr[Lap(2\tau/\varepsilon) \geq GS_Q/(3\varepsilon)] + \Pr[Lap(4\tau/\varepsilon) \geq 2GS_Q/(3\varepsilon)] \\
\leq &\frac{\beta}{GS_Q}
\end{aligned}$$

By a union bound, the SVT stops before $\tau = GS_Q/(6\log(GS_Q/\beta))$ with probability less than β. Above all, with probability at least $\frac{1}{2e} - \beta$, the truncation threshold selected is at least $GS_Q/(6\log(GS_Q/\beta))$. Denote E as the event $\tau \geq GS_Q/(6\log(GS_Q/\beta))$ and $\Pr[E] \geq \frac{1}{2e} - \beta$. Then, we have

$$\begin{aligned}
&\Pr[|Q(\mathsf{I},\tau) + Lap(\tau/\varepsilon) - Q(\mathsf{I})| \geq GS_Q/(6\varepsilon\log(GS_Q/\beta))|E] \\
\geq &\Pr[Q(\mathsf{I},\tau) + Lap(\tau/\varepsilon) \leq Q(\mathsf{I}) - GS_Q/(6\varepsilon\log(GS_Q/\beta))|E] \\
\geq &\Pr[Lap(\tau/\varepsilon) \leq -GS_Q/(6\varepsilon\log(GS_Q/\beta))|E] \qquad (13) \\
= &\frac{1}{2e} \qquad (14)
\end{aligned}$$

(13) is because, for any τ, $Q(\mathsf{I},\tau) \leq Q(\mathsf{I})$.

Above all,

$$\begin{aligned}
&\Pr[|Q(\mathsf{I},\tau) + Lap(\tau/\varepsilon) - Q(\mathsf{I})| \geq GS_Q/(6\varepsilon\log(GS_Q/\beta))] \\
\geq &\Pr[|Q(\mathsf{I},\tau) + Lap(\tau/\varepsilon) - Q(\mathsf{I})| \geq GS_Q/(6\varepsilon\log(GS_Q/\beta))|E] \times \Pr[E] \\
\geq &(\frac{1}{2e} - \beta)\frac{1}{2e}.
\end{aligned}$$

By setting $\beta = \frac{1}{4e}$, with probability at least $\frac{1}{8e^2}$, we have

$$|\mathcal{M}(\mathsf{I}) - Q(\mathsf{I})| \geq GS_Q/(6\varepsilon\log(4eGS_Q)).$$

Note that this analysis holds for every instance I, namely the mechanism in [37] adds the same amount of noise to all instances, which equals the worst-case noise (ignoring a logarithmic factor).