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

Wei Dong
Hong Kong University of Science and Technology
wdongac@cse.ust.hk

Juanru Fang
Hong Kong University of Science and Technology
jfangad@cse.ust.hk

Ke Yi
Hong Kong University of Science and Technology
yike@cse.ust.hk

Yuchao Tao
Duke University
yctao@cs.duke.edu

Ashwin Machanavajjhala
Duke University
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.

1. INTRODUCTION

Differential privacy (DP) 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. [14], who also formalized the *DP policy for relational databases with FK constraints*. The essence of their model (a rigorous definition is given in Section 2) 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 [13] 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 (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, 10]. 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 3 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 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 $\mathbf{I} = (I_1, I_2)$ 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 $\mathbf{I}'$ 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 $\mathbf{I}$ is $n\tau$, and that on $\mathbf{I}'$ 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 $\mathbf{R} = \{\texttt{Node}(\underline{\texttt{ID}}), \texttt{Edge}(\texttt{src}, \texttt{dst})\}$, Kasiviswanathan et al. [13] 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, 10] and self-join-free queries [17], 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 4) 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 $O(\log(GS_Q) \log\log(GS_Q) \cdot DS_Q(\mathbf{I}))$ for any instance $\mathbf{I}$ with constant probability. While we defer the formal definition of $DS_Q(\mathbf{I})$ to Section 3, what we can show is that it is an *per-instance lower bound*, i.e., any valid DP mechanism has to incur error $\Omega(DS_Q(\mathbf{I}))$ on $\mathbf{I}$ (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 [12], even for the mean estimation problem, i.e., the simple self-join-free query in Example 1.1. In practice, these log factors are usually between 10 to 100, and our experiments show that R2T has better utility than previous methods in most cases.

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 5), we extend the LP-based mechanism of [13], 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 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 7) using PostgreSQL and CPLEX. Experimental results (Section 8) 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. PRELIMINARIES

2.1 Database Queries

Let $\mathbf{R}$ be a database schema. We start with a multi-way join:

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

where $R_1, \ldots, R_n$ are relation names in $\mathbf{R}$ and each $\mathbf{x}_i$ 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 $\mathbf{x}_i \neq \mathbf{x}_j$, or one of the two atoms will be redundant. Let $var(J) := \mathbf{x}_1 \cup \cdots \cup \mathbf{x}_n$.

Let $\mathbf{I}$ be a database instance over $\mathbf{R}$. For any $R \in \mathbf{R}$, denote the corresponding relation instance in $\mathbf{I}$ as $\mathbf{I}(R)$. This is a *physical relation instance* of R. We use $\mathbf{I}(R, \mathbf{x})$ to denote $\mathbf{I}(R)$ after renaming its attributes to $\mathbf{x}$, 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 $J(\mathbf{I})$. More abstractly, let $\psi : \mathbf{dom}(var(J)) \to \mathbb{N}$ be a function that assigns non-negative integer weights to the join results, where $\mathbf{dom}(var(J))$ denotes the domain of $var(J)$. The result of evaluating Q on $\mathbf{I}$ is

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

Note that the function ψ only depends on the query. For a counting query, $\psi(\cdot) \equiv 1$; for an aggregation query, e.g. $\texttt{SUM}(A * B)$, $\psi(q)$ 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 $q \in J(\mathbf{I})$ does not satisfy the predicate, we simply set $\psi(q) = 0$.

Example 2.1. *Graph pattern counting queries can be formulated as SJA queries. Suppose we store a graph in a relational database by the schema* $\mathbf{R} = \{\texttt{Edge}(\texttt{src}, \texttt{dst}), \texttt{Node}(\underline{\texttt{ID}})\}$ *where* `src` *and* `dst` *are FKs referencing* `ID`*, then the number of length-3 paths can be counted by first computing the join*

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

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*)* $\texttt{A} \neq \texttt{C} \wedge \texttt{A} \neq \texttt{D} \wedge \texttt{B} \neq \texttt{D}$*. 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* $\texttt{A} < \texttt{D}$ *to eliminate the double counting.* □

2.2 Differential Privacy in Relational Databases with Foreign Key Constraints

We adopt the DP policy in [14], which defines neighboring instances by taking foreign key (FK) constraints into consideration. We model all the FK relationships as a directed acyclic graph over $\mathbf{R}$ by adding a directed edge from R to R' if R has an FK referencing the PK of R'. There is a. designated *primary private relation* R_P and any 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 $t_P \in \mathbf{I}(R_P)$ said to reference itself; (2) for $t_P \in \mathbf{I}(R_P)$, $t \in \mathbf{I}(R), t' \in \mathbf{I}(R')$, 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 $\mathbf{I}$ and $\mathbf{I}'$ are considered neighbors if $\mathbf{I}'$ can be obtained from $\mathbf{I}$ by deleting a set of tuples, all of which reference the same tuple $t_P \in \mathbf{I}(R_P)$, 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 $q \in J(\mathbf{I})$, we say that q references $t_P \in \mathbf{I}(R_P)$ if $|t_P \bowtie q| = 1$.

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

Example 2.2. *Consider the TPC-H schema:*

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

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 2.1 is such an example. Following [14], 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 2.1, we add $\texttt{Node}(\texttt{A})$, $\texttt{Node}(\texttt{B})$, $\texttt{Node}(\texttt{C})$, and $\texttt{Node}(\texttt{D})$.

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 2.1, 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 $\mathbf{I}, \mathbf{I}'$, and any output y, we have

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

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

3. INSTANCE OPTIMALITY OF DP MECHANISMS WITH FK CONSTRAINTS

Global sensitivity and worst-case optimality.

The standard DP mechanism is the Laplace mechanism [9], 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 $GS_Q = \max_{\mathbf{I} \sim \mathbf{I}'} |Q(\mathbf{I}) - Q(\mathbf{I}')|$ is the

For most parts of the paper, we consider the case where there is only one *primary private relation* in $\mathbf{R}$; the case with multiple primary private relations can be transformed to a case with single primary private relation [6]

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, 10, 17], we restrict to a set of instances $\mathcal{I}$ such that

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

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 $\mathbf{I}$ and an SJA query Q, for a tuple $t_P \in \mathbf{I}(R_P)$, its *sensitivity* is

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

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

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

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

$$Q(\mathbf{I}) = \sum_{t_P\in R_P} S_Q(\mathbf{I}, t_P),$$

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 $\mathbf{I}(R_P)$ may affect multiple $S_Q(\mathbf{I}, t_P)$'s, making the neighboring relationship more complicated than in the sum estimation problem, where two neighboring instances differ by only one datum [2, 10].

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

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

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

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

for every $\mathbf{I}$, where c is called the *optimality ratio*. Unfortunately, for any $\mathbf{I}$, one can design a trivial $M'(\cdot) \equiv Q(\mathbf{I})$ that has 0 error on $\mathbf{I}$ (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 [9].

To avoid such a trivial M', [3, 8] consider a relaxed version of instance optimality where we compare M against any M' that is required to work well not just on $\mathbf{I}$, but also on its neighbors, i.e., we raise the target error from $\mathcal{L}_{\text{ins}}(\mathbf{I})$ to

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

Vadhan [18] observes that $\mathcal{L}_{\text{nbr}}(\mathbf{I}) \ge LS_Q(\mathbf{I})/2$, where

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

is the *local sensitivity* of Q at $\mathbf{I}$. This instance optimality has been used for certain machine learning problems [3] and conjunctive queries without FKs [8]. However, it has an issue for SJA queries in a database with FK constraints: For any $\mathbf{I}$, we can add a t_P to $\mathbf{I}(R_P)$ together with tuples in the secondary private relations all referencing t_P, obtaining an $\mathbf{I}'$ such that $S_Q(\mathbf{I}', t_P) = GS_Q$, 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 [16, 11, 7, 8], 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 $\mathbf{I}'$ of $\mathbf{I}$. Under the neighborhood definition with FK constraints, this means that $\mathbf{I}'$ can be any instance obtained from $\mathbf{I}$ 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, [10] 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

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

namely, we require M' to work well only on $\mathbf{I}'$ and its *down-neighbors*, which can be obtained only by removing a tuple t_P already in $\mathbf{I}(R_P)$ 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 [3, 8, 10]. Note that for such an instance-optimal M (by our definition), there still exist $\mathbf{I}, M'$ such that M' does better on $\mathbf{I}$ than M, but if this happens, M' must do worse on one of the down-neighbors of $\mathbf{I}$, which is as typical as $\mathbf{I}$ itself.

Using the same argument from [18], we have $\mathcal{L}_{\text{d-nbr}}(\mathbf{I}) \ge DS_Q(\mathbf{I})/2$, where

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

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

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

4. R2T: INSTANCE-OPTIMAL TRUNCATION

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

(1) For any τ, the global sensitivity of $Q(\cdot, \tau)$ is at most τ.

(2) For any τ, $Q(\mathbf{I}, \tau) \le Q(\mathbf{I})$.

(3) For any $\mathbf{I}$, there exists a non-negative integer $\tau^*(\mathbf{I}) \le GS_Q$ such that for any $\tau \ge \tau^*(\mathbf{I})$, $Q(\mathbf{I}, \tau) = Q(\mathbf{I})$.

We describe various choices for $Q(\mathbf{I}, \tau)$ depending on the DP policy and whether the query contains self-joins and/or projections in the subsequent sections. Intuitively, such a $Q(\mathbf{I}, \tau)$ gives a stable (property (1)) underestimate (property (2)) of $Q(\mathbf{I})$, while reaches $Q(\mathbf{I})$ for a sufficiently large τ (property (3)). Note that $Q(\mathbf{I}, \tau)$ 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 $Q(\mathbf{I}, \tau)$, R2T is works as follows. For a probability β, we first compute

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

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

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

The privacy of R2T is straightforward: Since $Q(\mathbf{I}, \tau^{(j)})$ has global sensitivity at most $\tau^{(j)}$, and the third term of (7) is independent of $\mathbf{I}$, each $\tilde{Q}(\mathbf{I}, \tau^{(j)})$ satisfies $\frac{\varepsilon}{\log(GS_Q)}$-DP by the standard Laplace mechanism. Collectively, all the $\tilde{Q}(\mathbf{I}, \tau^{(j)})$'s satisfy ε-DP by the basic composition theorem [9]. Finally, returning the maximum preserves DP by the post-processing property of DP.

Utility analysis.

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

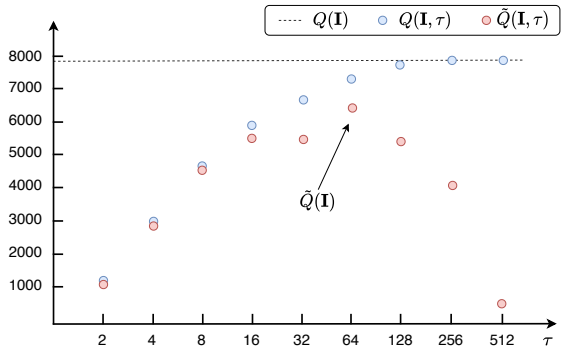

Figure 1: An illustration of R2T.

THEOREM 1. *On any instance* $\mathbf{I}$*, with probability at least* $1 - \beta$*, we have*

$$Q(\mathbf{I}) - 4\log(GS_Q)\ln\left(\frac{\log(GS_Q)}{\beta}\right)\frac{\tau^*(\mathbf{I})}{\varepsilon} \le \tilde{Q}(\mathbf{I}) \le Q(\mathbf{I}).$$

5. TRUNCATION FOR SJA QUERIES

In this section, we will design a $Q(\mathbf{I}, \tau)$ with $\tau^*(\mathbf{I}) = DS_Q(\mathbf{I})$ for SJA queries. Plugged into Theorem 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 $q \in J(\mathbf{I})$ 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 $S_Q(\mathbf{I}, t_P) > \tau$ and then summing up the rest) is a valid $Q(\mathbf{I}, \tau)$ that satisfies the 3 properties required by R2T with $\tau^*(\mathbf{I}) = DS_Q(\mathbf{I})$.

When there are self-joins, naive truncation does not satisfy property (1), as illustrated in Example 1.2, where all $S_Q(\mathbf{I}, t_P)$'s in two neighboring instances may differ. Below we generalize the LP-based mechanism for graph pattern counting [13] to arbitrary SJA queries, and show that it satisfies the 3 properties with $\tau^*(\mathbf{I}) = DS_Q(\mathbf{I})$.

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

$$C_j(\mathbf{I}) := \{k : q_k(\mathbf{I}) \text{ references } t_j(\mathbf{I})\}. \quad (9)$$

For each $k \in [|J(\mathbf{I})|]$, introduce a variable u_k, which represents the weight assigned to the join result $q_k(\mathbf{I})$. We return

The probability β only concerns about the utility but not privacy.

log has base 2 and ln has base e.

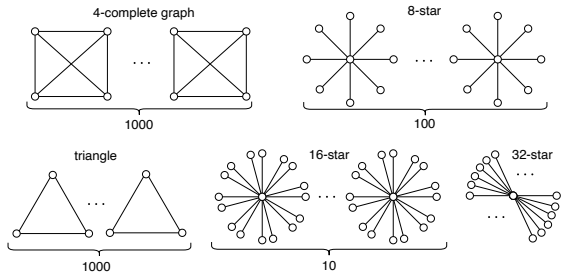

Figure 2: Example of edge counting.

the optimal solution of the following LP as $Q(\mathbf{I}, \tau)$:

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

LEMMA 1. *For SJA queries, the $Q(\mathbf{I}, \tau)$ defined above satisfies the 3 properties required by R2T with $\tau^*(\mathbf{I}) = DS_Q(\mathbf{I})$.*

EXAMPLE 5.1. *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*

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

on the graph data schema introduced in Example 2.1. 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 $\mathbf{I} \in \mathcal{I}$ is 256. We set $\beta = 0.1$ and $\varepsilon = 1$.

Now, suppose we are given an $\mathbf{I}$ 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

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

We run R2T with $\tau^{(j)} = 2^j$ for $j = 1, \ldots, 8$. 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 $\texttt{ID1} < \texttt{ID2}$. To calculate $Q(\mathbf{I}, \tau)$, 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

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

In addition, we have $Q(\mathbf{I}, 0) = 0$ and $Q(\mathbf{I}, \tau) = 9992$ for $\tau \ge 32$.

Then, let's see how to run R2T with these $Q(\mathbf{I}, \tau)$'s. Let $\epsilon = 1$, $\beta = 0.1$, and $GS = 2^{10}$. Besides, for convenience, assume $Lap(1)$ returns -1 and 1 by turns. Plugging these into (7), we have

$$\begin{aligned} \tilde{Q}(\mathbf{I},2) =& 7222 + (-1)\cdot 20 - 92.1 = 7110 \\ \tilde{Q}(\mathbf{I},4) =& 9444 + 1\cdot 40 - 184 = 9300 \\ \tilde{Q}(\mathbf{I},8) =& 9888 + (-1)\cdot 80 - 368 = 9440 \\ \tilde{Q}(\mathbf{I},16) =& 9976 + 1\cdot 160 - 737 = 9399 \\ \tilde{Q}(\mathbf{I},32) =& 9992 + (-1)\cdot 320 - 1474 = 8198 \\ \tilde{Q}(\mathbf{I},64) =& 9992 + 1\cdot 640 - 2947 = 7685 \\ & \cdots \end{aligned}$$

Finally, with (8), we have $\tilde{Q}(\mathbf{I}) = \tilde{Q}(\mathbf{I}, 8) = 9440$.

□

6. TRUNCATION FOR SPJA QUERIES

A projection reduces the query answer, hence its sensitivity, so it requires less noise. However, it makes achieving instance optimality harder: even in the simple case $|\pi_{x_2}(R_1(x_1) \bowtie R_2(x_1, x_2))|$, it is impossible to achieve the error $f(\mathbf{I}) \cdot DS_Q(\mathbf{I})$ at each instance $\mathbf{I}$ for any function $f(\mathbf{I})$. To address this issue, we propose a truncation for SPJA queries with error depending on another instance specific notation. Please read our full version paper [6] for more details.

7. 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 $\psi(q_k(\mathbf{I}))$ for each join result, as well as $C_j(\mathbf{I})$, which stores the referencing relationships between tuples in $\mathbf{I}(R_P)$ and $J(\mathbf{I})$.

EXAMPLE 7.1. *Suppose we use the TPC-H schema (shown in Figure 4), where we designate* `Supplier` *and* `Customer` *as*

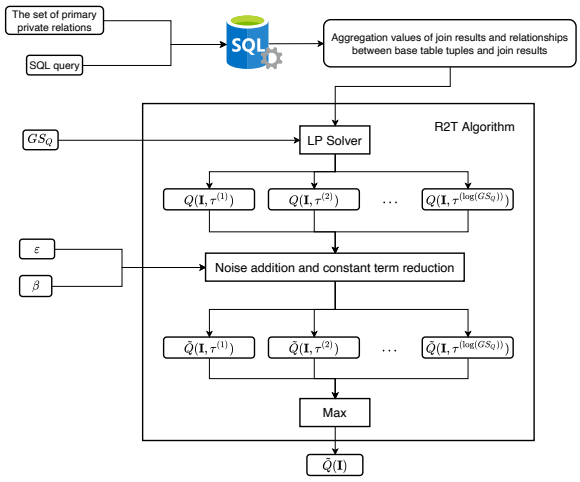

Figure 3: System structure.

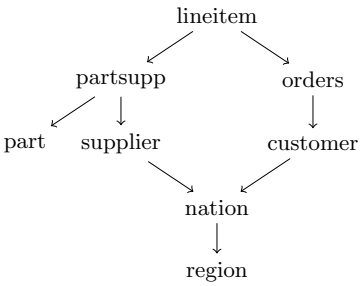

Figure 4: The foreign-key graph of TPC-H schema.

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 and we present another effective technique to speed up the process in our full version paper [6].

8. EXPERIMENTS

We conducted experiments on graph pattern counting queries under node-DP, an important special case of the SPJA queries with FK constraints. Here, we compare R2T with naive truncation with smooth sensitivity (NT) [13], smooth distance estimator (SDE) [4], recursive mechanism (RM) [5], and the LP-based mechanism (LP) [13]. We also implement some experiments on general SPJA queries to compare R2T with the local sensitivity-based mechanism (LS) [17]. The experimental results show R2T achieves order-of-magnitude improvements over LS in terms of utility, with similar running times. For the space limit, we move this part to our full version paper [6].

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.

8.1 Setup

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 $Q_{\square}$. 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 [15]. 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 $GS_{Q_{\square}} = D^3$.

The LP mechanism requires a truncation threshold τ, but [13] 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.

All experiments were conducted on a Linux server with a 24-core 2.2GHz Intel Xeon CPU and 256GB of memory.

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.

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 $\varepsilon = 0.8$.

8.2 Experimental Results

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 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 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. We also conducted experiments to see how the privacy parameter ε affects various mechanisms [6]. The result shows R2T has a high utility even for a small ε.

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 [13] 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.

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.

9. 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.

10. REFERENCES

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

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

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

[4] J. Blocki, A. Blum, A. Datta, and O. 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.

[5] S. Chen and S. 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.

[6] W. Dong, J. Fang, K. Yi, Y. Tao, and A. Machanavajjhala. R2t: Instance-optimal truncation for differentially private query evaluation with foreign keys. url=https://www.cse.ust.hk/ yike/R2T.pdf, 2022.

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

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

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

[10] Z. Huang, Y. Liang, and K. Yi. Instance-optimal mean estimation under differential privacy. In *NeurIPS*, 2021.

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

[12] V. Karwa and S. Vadhan. Finite sample differentially private confidence intervals. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[13] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography Conference*, pages 457–476. Springer, 2013.

[14] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment*, 12(11):1371–1384, 2019.

[15] J. Leskovec and A. Krevl. Snap datasets: Stanford large network dataset collection (2014). *URL http://snap. stanford. edu/data*, page 49, 2016.

[16] K. Nissim, S. Raskhodnikova, and A. 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.

[17] Y. Tao, X. He, A. Machanavajjhala, and S. 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.

[18] S. Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017.