Residual Sensitivity for Differentially Private Multi-Way Joins

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

ABSTRACT

A general-purpose query engine that supports a large class of SQLs under differential privacy is the holy grail in privacy-preserving query release. The join operator presents a major difficulty towards realizing this goal, since a single tuple may affect a large number of query results, and the problem worsens as more relations are involved in the join. The traditional approach of global sensitivity fails to work as it assumes pessimistically that every pair of tuples from two different relations may join. To address the issue, instance-dependent sensitivity measures have been proposed, but so far none has met the following three desiderata for it to be truly practical: (1) the released answer should have low noise levels (i.e., high utility); (2) it can be computed efficiently; and (3) the method can be easily integrated into an existing relational database. This paper presents the first differentially private mechanism for multi-way joins that satisfies all three desiderata while supporting any number of private relations, moving us one step closer to a full-featured query engine for private relational data.

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, counting query, join

ACM Reference Format:
Wei Dong Ke Yi. 2021. Residual Sensitivity for Differentially Private Multi-Way Joins. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 18–27, 2021, Virtual Event, China.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

Suppose a data analyst is interested in the total number of items sold this year where the customer and supplier are from the same nation in a given region, s/he would issue the following query (assuming the TPC-H schema):

```
SELECT count(*)
FROM Region, Nation, Customer, Orders, Supplier, Lineitem
WHERE Orders.Orderdate > 2020-01-01
  AND Region.Name = '[REGION]'
 AND Region.RK = Nation.RK AND Lineitem.SK = Supplier.SK
 AND Nation.NK = Customer.NK AND Customer.CK = Orders.CK
 AND Orders.OK = Lineitem.OK AND Nation.NK = Supplier.NK
```

Such queries involving multi-way joins are very common in today's data analytical tasks, and are a central problem in databases that have been extensively studied in the literature. Sophisticated query processing algorithms and systems have been and are continually being developed and optimized throughout the years.

However, many datasets contain private information, and privacy concerns have become the main hurdle to making use of today's big data for knowledge discovery and decision making. In this example, while it might be safe to reveal `Region` and `Nation`, the other four relations contain private relationship information between various entities, such as which customer has placed a particular order, which items are contained in an order, which suppliers provide a certain item, so their privacy must be protected.

This paper studies the problem of how to release a multi-way join counting query under *differential privacy (DP)* [9], which provides a formal privacy guarantee to individual tuples in the database. While deferring the details to Section 3, a dominating approach is the following *sensitivity* framework.

(1) Compute the query answer on the given database instance.
(2) Compute some notions of sensitivity of the query, which measure the difference between the query answers on two database instances that differ by one tuple (a.k.a. *neighboring* database instances).
(3) Release a noise-masked query answer, where the noise is drawn from some zero-mean distribution, calibrated appropriately according to the sensitivity.

Step (1) and (3) are both well understood in the literature, so the key challenge is step (2). There are three desiderata when designing a sensitivity measure, by order of decreasing importance:

(1) Utility: The sensitivity should be as small as possible, as the noise level is proportional. Adding noise with a magnitude greater than the actual query answer would completely destroy its utility.
(2) Efficiency: The sensitivity should be computed efficiently.
(3) Integration: It can be easily integrated into any existing relational database system.

Below, we review previous measures of sensitivity for multi-way joins along the three desiderata, before presenting our proposal.

1.1 Previous Work

Global sensitivity. The *global sensitivity* is defined as the maximum difference between the query answers on *any* two neighboring database instances. Note that global sensitivity is a property of the

SIGMOD '21, June 18–27, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

query only, and does not depend on the actual instance. Equivalently speaking, global sensitivity considers the worst-case instance, and measures the amount of change in the query answer when a tuple is changed in that worst-case instance. This works well for queries without joins, where the global sensitivity is just 1. However, the global sensitivity becomes unbounded when (unrestricted) joins are present.

Local and smooth sensitivity. When the global sensitivity is high or unbounded, it is tempting to use the sensitivity of the query on the particular given instance, which is usually much lower, except on contrived instances. This is referred to as the *local sensitivity*. However, as pointed out by Nissim et al. [21], using the local sensitivity to calibrate noise is not differentially private. This is because the local sensitivity can be very different on two neighboring databases, so the noise level may reveal information about an individual tuple. Essentially, the problem is that local sensitivity, when considered as a query, has high global sensitivity. To get around the problem, the idea is to use a smooth (i.e., having low global sensitivity) upper bound of the local sensitivity. Clearly, the smaller this upper bound is, the better utility we would obtain, and the tightest smooth upper bound is termed the *smooth sensitivity* [21]. Please see Section 3 for more precise definitions.

Although smooth sensitivity has met desideratum (1), (2) and (3) are less clear. In fact, it is shown that for certain problems, computing or even approximating the smooth sensitivity is NP-hard [21]. While the computational hardness of smooth sensitivity of multi-way joins is still open, we have recently found an $N^{O(\log N)}$-time[1] algorithm (see Appendix A), which is actually an indication that it may not be NP-hard. Nevertheless, a polynomial-time algorithm still remains elusive, and even if there is one, it probably will be very complicated, thus making desideratum (3) hard to achieve.

Elastic sensitivity. Due to the lack of an efficient algorithm for computing the smooth sensitivity for multi-way joins, Johnson et al. [13] proposed *elastic sensitivity*, which is also a smooth upper bound of local sensitivity, but not as tight as smooth sensitivity. Elastic sensitivity achieves both (2) and (3): it can be computed in linear time, and can be implemented easily using a constant number of SQL queries, plus a user-defined function (UDF) that combines the answers of these queries using a certain formula. However, it does not really achieve (1). Theoretically, the gap between elastic sensitivity and smooth sensitivity can be as large as $O\left(N^{n-1}\right)$ for an n-way join on a database containing N tuples. In practice, it often yields noise levels that are orders-of-magnitude higher than the actual query answer, as demonstrated in the experiments of [13], as well as our own experiments.

1.2 Our Proposal: Residual Sensitivity

This paper proposes *residual sensitivity* for multi-way join counting queries without self-joins. It is also a smooth upper bound of local sensitivity, thus can be used to calibrate noise to ensure differential privacy. More importantly, we show that it meets all three desiderata outlined earlier:

(1) Utility: Theoretically, we prove that residual sensitivity is always smaller than elastic sensitivity on any query and any database instance. More importantly, we prove that residual sensitivity is at most a constant-factor larger than smooth sensitivity. Thus, it can also be considered as a constant-factor approximation of smooth sensitivity, for which no polynomial-time algorithms are known. Empirically, we demonstrate that residual sensitivity is much smaller than elastic sensitivity on a collection of queries using both benchmark and real-world datasets, while the gap can go up to 7 orders of magnitude. When applied to noise calibration, this makes the noise-masked answers of many queries usable under stricter privacy requirements, especially for cyclic queries and/or over skewed data.
(2) Efficiency: In theory, we show that residual sensitivity can be computed in polynomial time, where the exponent depends only on the join structure, not the database. In practice, the running time appears to be linear, and is only slower than that for computing elastic sensitivity by a small constant factor.
(3) Integration: Similar to elastic sensitivity, residual sensitivity can also be computed by executing a constant number of SQL queries, and then combining their answers using a UDF, although the queries are slightly more complicated than those for elastic sensitivity. We have built a system prototype based on PostgreSQL that can automatically computes a differentially private answer given any SQL query consisting of joins, selections, projections, and group-by.

The paper is organized as follows. After reviewing other related work in Section 2, we begin the technical development in Section 3. Section 4.5 defines residual sensitivity and describes how it can be computed. Section 5 gives its theoretical analysis while Section 6 presents the experimental results.

2 RELATED WORK

Since its introduction [8, 9], DP has attracted a lot of interest in academia, government agencies, and industry. As mentioned, if the query doesn't have joins, then the global sensitivity is 1, so the query answer can be released by just adding $O(1)$ noise. Therefore, most efforts along this direction have been devoted to studying how to release many query answers (with different selection conditions) under a given privacy budget [3, 6, 11, 24, 25, 28, 30], in a way better than direct composition [10].

The problem becomes much more challenging when joins are present, because a single tuple may now affect many join results. A relatively easy approach is to add constraints so as to reduce the global sensitivity. McSherry [19] solves the problem by only restricting to one-to-one joins. Proserpio et al. [23] propose wPINQ to extend the work of McSherry to support general equijoins: by assigning weights to tuples and scaling down the weights, their algorithm ensures each tuple can at most affect one on final counting result. However, this only works well when one tuple affects a fixed number of results. Palamidessi and Stronati [22] add constraints on the attribute range. Arapinis et al. [2] and Narayan et al. [20] consider functional dependencies and cardinality constraints. In

[1]Throughout the paper, we use *data complexity* [1] when talking about running times, i.e., the running time is measured as a function of the database size N, while the query size (i.e., number of relations and attributes in the query) is considered a constant.

contrast, elastic sensitivity and residual sensitivity do not require any constraints on the join structure or the database.

In the relational model, two DP policies have been proposed and studied, depending on whether foreign key constraints are considered when defining neighboring instances. Our DP policy, which is the same as in [13, 16, 20, 22, 23], does not consider foreign key constraints, and two database instances are neighboring if they differ by one tuple in any relation. In the DP policy considering foreign key constraints [17, 27], two database instances are neighboring if one can be obtained from the other by deleting a tuple t in the *primary private relation* and all tuples in the other relations that can join with t via foreign key constraints. The two DP policies are incomparable in terms of their privacy guarantee: On the one hand, the DP policy considering foreign key constraints offers stronger privacy guarantee for the primary private relation, but it only supports one such relation. On the other hand, the DP policy not considering foreign key constraints provides relationship-level guarantees to any number of relations. Which DP policy to use will depend on the level of privacy needed by the application. If the relationships, such as friendships, seller-buyer relationships, caller-callee relationships, are sensitive, then the former should be adopted. If the entities (including all information about the entity reachable via foreign key constraints) are to be protected, then the latter is more appropriate.

There have also been a series of works on graph pattern counting queries under DP [4, 5, 7, 14, 15, 26, 29]. Their work differs from ours in two aspects: First, by storing all edges in a two-column relation, a graph pattern query can be formulated as a multi-way self-join on one relation of arity 2. But since we cannot handle self-joins (yet), we cannot deal with such queries. Meanwhile, they cannot handle multi-way joins, either, which may involve any number of relations of arbitrary arity. Second, these methods have efficient algorithms only for specific graph patterns, such as triangles, stars, and cliques; otherwise, they will suffer from high computation cost. On the other hand, elastic/residual sensitivity can be computed in polynomial time on any join structure.

3 PRELIMINARIES

3.1 Multi-way Joins

Let $\mathbf{R}$ be a database schema that consists of a set of relation names. We consider counting queries defined by multi-way (natural) joins of the form

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

where $R_1, \ldots, R_n \in \mathbf{R}$, and $\mathbf{x}_i$ are the attributes of R_i. We call each $R_i(\mathbf{x}_i)$ an *atom*. In this paper, we consider self-join-free queries, namely, all the R_i's are distinct.

Let $\mathbf{I}$ be a database instance. For a relation name $R \in \mathbf{R}$, let $\mathbf{I}(R)$ denote the instance of relation R. Given a multi-way join query q defined as above, we use I_i as a shorthand for $\mathbf{I}(R_i)$. Let $N = |\mathbf{I}|$ be the input size, and denote the query results of q on $\mathbf{I}$ as $q(\mathbf{I})$. A counting query returns the cardinality of the query results, denoted $|q(\mathbf{I})|$.

For an attribute x, we use $\mathbf{dom}(x)$ to denote the domain of x. For $\mathbf{x} = (x_1, \ldots, x_k)$, let $\mathbf{dom}(\mathbf{x}) = \mathbf{dom}(x_1) \times \cdots \times \mathbf{dom}(x_k)$. We use $[n]$ to denote $\{1, \ldots, n\}$, and $[i, j] = \{i, \ldots, j\}$. Per convention, define $[n] = \emptyset$ if $n \le 0$, and $[i, j] = \emptyset$ if $i > j$. For any $E \subseteq [n]$, define $\bar{E} = [n] - E$.

3.2 Differential Privacy

For two relation instances I, I', we use $d(I, I')$ to denote the distance between I and I', which is the minimum number of steps to change I into I', where each step is insertion, deletion, or change of a tuple. The distance between two database instances is thus $d(\mathbf{I}, \mathbf{I}') = \sum_{i \in [n]} d(I_i, I_i')$. We use $P \subseteq [n]$ to denote the set of private relations and let $n_P = |P|$. Two instances can only differ in the private relations, i.e., $d(I_i, I_i') = 0$ for every $i \in \bar{P}$. If $d(\mathbf{I}, \mathbf{I}') = 1$, we call $\mathbf{I}, \mathbf{I}'$ *neighboring instances*.

We allow P to be any subset of relations in the query q. There are many scenarios where multiple relations are private. For example, in social networks and knowledge graphs, edges have different types such as "friendship", "coauthorship", "is a member of", "belongs to", and they modeled as different relations. In a more traditional relational schema like the TPC-H, different relations store different relationships between customers and orders, orders and suppliers, parts and suppliers, etc.

Definition 3.1 (Differential privacy). For $\varepsilon, \delta > 0$, an algorithm $\mathcal{M}$ is (ε, δ)-differentially private if for any neighboring instances $\mathbf{I}, \mathbf{I}'$ and any subset of outputs Y,

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

Typically, ε is a constant ranging from 0.1 to 10, with smaller values corresponding to stronger privacy guarantees. On the other hand, δ should be much smaller than $1/N$ to ensure the privacy of individual tuples; in particular, the case where $\delta = 0$ is referred to as *pure differential privacy*, which is more desirable if achieved.

The most common technique of achieving differential privacy is to mask the query result by adding random noise drawn from a certain zero-mean probability distribution. The noise level (i.e., the standard deviation of the distribution) should depend on the difference between the query results on $\mathbf{I}$ and $\mathbf{I}'$, which is captured by the notion of *sensitivity*. The *local sensitivity* of q at instance $\mathbf{I}$ is how much $|q|$ can change at most if one tuple in $\mathbf{I}$ is changed, i.e.,

$$LS_q(\mathbf{I}) = \max_{\mathbf{I}', d(\mathbf{I}, \mathbf{I}') = 1} \big||q(\mathbf{I})| - |q(\mathbf{I}')|\big|. \tag{1}$$

The *global sensitivity* of q is

$$GS_q = \max_{\mathbf{I}} LS_q(\mathbf{I}).$$

It is well known that adding noise proportional to the global sensitivity ensures privacy, but unfortunately, the global sensitivity of many queries can be very high as the max is taken over all instances $\mathbf{I}$. For an n-way join, the global sensitivity can be as high as $O(N^{n-1})$, which is unbounded as N is considered unbounded and private[2]. On the other hand, the local sensitivity is often much smaller in most real-world instances. However, as observed in [21], local sensitivity cannot be used to scale the noise directly, since $LS_q(\mathbf{I})$ and $LS_q(\mathbf{I}')$ can differ a lot on two neighboring instances $\mathbf{I}$ and $\mathbf{I}'$. Very different amounts of noise would be added to $q(\mathbf{I})$ and $q(\mathbf{I}')$, which breaches privacy.

[2]In some DP definitions, the instance size N is consider public information. In this case, the global sensitivity is bounded, but still very high.

3.3 Smooth Sensitivity

To address the issue, Nissim et al. [21] proposed *smooth sensitivity*. Similar with local sensitivity, smooth sensitivity is also instance-dependent and usually much smaller than global sensitivity. But different from local sensitivity, it eliminates abrupt changes between neighboring instances, hence the name "smooth sensitivity".

The smooth sensitivity is based on the *local sensitivity at distance* k, $LS_q^{(k)}$, which is defined as

$$LS_q^{(k)}(\mathbf{I}) = \max_{\mathbf{I}', d(\mathbf{I},\mathbf{I}') \le k} LS_q(\mathbf{I}').$$

Note that $LS_q^{(0)}(\mathbf{I}) = LS_q(\mathbf{I})$ and $\forall k \ge 0, LS_q^{(k)}(\mathbf{I}) \le GS_q$. In fact, $LS_q^{(k)}$ can be equivalently defined as

$$LS_q^{(k)}(\mathbf{I}) = \max_{\mathbf{I}', d(\mathbf{I},\mathbf{I}')=k} LS_q(\mathbf{I}') \tag{2}$$

if dummy tuples are allowed in database.

Definition 3.2. The *β-smooth sensitivity* of q is

$$SS_q(\mathbf{I}) = \max_{k \ge 0} e^{-\beta k} LS_q^{(k)}(\mathbf{I}). \tag{3}$$

An important property of $LS_q^{(k)}$ is that for any $\mathbf{I}, \mathbf{I}'$ such that $d(\mathbf{I}, \mathbf{I}') = 1$, and any $k \ge 0$, $LS_q^{(k)}(\mathbf{I}) \le LS_q^{(k+1)}(\mathbf{I}')$. This ensures the "smoothness" of $SS_q(\cdot)$: $SS_q(\mathbf{I})$ and $SS_q(\mathbf{I}')$ differ by at most a constant factor e^β on any two neighboring instances $\mathbf{I}$ and $\mathbf{I}'$.

Computing the smooth sensitivity by definition in general takes exponential time. Indeed, it has been shown that it is NP-hard to compute the smooth sensitivity for certain functions [21]. By exploiting special properties of the problem at hand, the running time can be reduced to polynomial; examples include the median, minimum spanning tree, and triangle counting [21]. However, it is still an open problem whether the smooth sensitivity of multi-way joins can be computed in polynomial time. In Appendix A, we describe an algorithm with running time $N^{O(\log N)}$. Such a running time is said to be *quasi-polynomial*, which is super-polynomial but sub-exponential. This suggests that computing the smooth sensitivity for multi-way joins may not be NP-hard, although a polynomial-time algorithm still remains elusive.

3.4 Smooth Upper Bound on Local Sensitivity

Fortunately, we do not have to compute the smooth sensitivity exactly. Nissim et al. [21] showed that any smooth upper bound $\hat{SS}_q(\mathbf{I})$ of local sensitivity can be used for privacy-preserving query publishing, which is also the focus of this paper.

Definition 3.3 (Smooth upper bound on local sensitivity). For $\beta > 0$, a *β-smooth upper bound on local sensitivity*, or simply *smooth upper bound*, is any function $\hat{SS}_q(\cdot)$ such that

(1) $\forall \mathbf{I}, \hat{SS}(\mathbf{I}) \ge LS_q(\mathbf{I})$; and
(2) $\forall \mathbf{I}, \mathbf{I}'$ such that $d(\mathbf{I}, \mathbf{I}') = 1$, $\hat{SS}_q(\mathbf{I}) \le e^\beta \cdot \hat{SS}_q(\mathbf{I}')$.

Note that setting $\hat{SS}_q(\mathbf{I}) = GS_q$ for all $\mathbf{I}$ certainly meets the two conditions, but it is a very loose upper bound. On the other hand, it has been shown that the first condition can be replaced by $\hat{SS}_q(\mathbf{I}) \ge SS_q(\mathbf{I})$, meaning that the smooth sensitivity $SS_q(\cdot)$ is actually the smallest smooth upper bound $\hat{SS}_q(\cdot)$ that one can take. Therefore, the problem is how to compute, in polynomial time, an $\hat{SS}_q(\cdot)$ that is as close to $SS_q(\cdot)$ as possible. This paper gives essentially the best possible answer to this problem: We show how to compute an $\hat{SS}_q(\mathbf{I})$ in polynomial time that is at most a constant factor larger than $SS_q(\mathbf{I})$, for an arbitrary multi-way join q over any instance $\mathbf{I}$.

Nissim et al. [21] proposed two methods for computing a smooth upper bound (Claim 3.2 and 3.3 in [21]). They utilize GS_q, the global sensitivity, and $\hat{LS}_q^{(k)}$, an upper bound on $LS_q^{(k)}$, respectively. We observe that their two methods can actually be combined to make use of both GS_q and $\hat{LS}_q^{(k)}$, leading to the following unified approach to obtaining a smooth upper bound:

Theorem 3.4. *Let*

$$\hat{SS}_q(\mathbf{I}) = \max_{k \ge 0} \left(e^{-\beta k} \min(\hat{LS}_q^{(k)}(\mathbf{I}), \hat{GS}_q) \right), \tag{4}$$

where

(1) $LS_q^{(k)}(\mathbf{I}) \le \hat{LS}_q^{(k)}(\mathbf{I})$ *for any* k *and* $\mathbf{I}$*;*
(2) $\hat{LS}_q^{(k)}(\mathbf{I}) \le \hat{LS}_q^{(k+1)}(\mathbf{I}')$ *for any* k *and any* $\mathbf{I}, \mathbf{I}'$ *such that* $d(\mathbf{I}, \mathbf{I}') = 1$*; and*
(3) $GS_q \le \hat{GS}_q$.

Then $\hat{SS}_q(\cdot)$ is a β-smooth upper bound on local sensitivity.

Proof. We prove the two conditions required by a smooth upper bound in Definition 3.3. Condition (1) is obvious, since $\hat{SS}_q(\mathbf{I}) \ge \min(\hat{LS}_q^{(0)}(\mathbf{I}), \hat{GS}_q)$, while both $\hat{LS}_q^{(0)}(\mathbf{I})$ and $\hat{GS}_q$ are upper bounds of $LS_q(\mathbf{I})$.

For condition (2), consider any $\mathbf{I}, \mathbf{I}'$ with $d(\mathbf{I}, \mathbf{I}') = 1$. From the requirement on $\hat{LS}_q^{(k)}$, we have

$$\min\left(\hat{LS}_q^{(k)}(\mathbf{I}), \hat{GS}_q\right) \le \min\left(\hat{LS}_q^{(k+1)}(\mathbf{I}'), \hat{GS}_q\right).$$

Define

$$k^{\mathbf{I}} = \arg\max_k \left(e^{-\beta k} \min(\hat{LS}_q^{(k)}(\mathbf{I}), \hat{GS}_q) \right).$$

Then

$$\begin{aligned}
\hat{SS}_q(\mathbf{I}) &= e^{-\beta k^{\mathbf{I}}} \min\left(\hat{LS}_q^{(k^{\mathbf{I}})}(\mathbf{I}), \hat{GS}_q\right) \\
&\le e^\beta e^{-\beta(k^{\mathbf{I}}+1)} \min\left(\hat{LS}_q^{(k^{\mathbf{I}}+1)}(\mathbf{I}'), \hat{GS}_q\right) \\
&\le e^\beta \max_k \left(e^{-\beta k} \min(\hat{LS}_q^{(k)}(\mathbf{I}'), \hat{GS}_q \right) = e^\beta \hat{SS}_q(\mathbf{I}'). \qquad \square
\end{aligned}$$

Remark 1. Note that in the $\max_{k \ge 0}$ of (4), it suffices to consider k up to $\hat{k} = \frac{\ln(\hat{GS}_q)}{\beta}$: For any $k > \hat{k}$, we have

$$e^{-\beta k} \min\left(\hat{LS}_q^{(k)}(\mathbf{I}), \hat{GS}_q\right) \le e^{-\beta k} \hat{GS}_q < 1,$$

so it cannot possibly yield the maximum.

Remark 2. Condition (1) in Theorem 3.4 can be equivalently changed to (1′) $LS_q(\mathbf{I}) \le \hat{LS}_q^{(0)}(\mathbf{I})$. To see why, note that (1) $\Rightarrow$ (1′) trivially. For the other direction, we have (1′) + (2) $\Rightarrow$ (1) by a simple induction proof, where the key is to use $LS_q^{(k+1)}(\mathbf{I}) = \max_{\mathbf{I}', d(\mathbf{I},\mathbf{I}')=1} LS_q^{(k)}(\mathbf{I}')$. We keep the seemingly more restrictive condition (2) in Theorem 3.4, because it actually gives us more

intuition on how to design an $\hat{LS}_q^{(k)}(\mathbf{I})$, as will be seen in our development of residual sensitivity for multi-way joins.

Remark 3. If GS_q is unbounded, we set $\hat{GS}_q = \infty$. In this case, $\hat{SS}_q(\mathbf{I})$ may be unbounded. However, for most natural problems (including multi-way joins; see Lemma 4.12), one can show that $e^{-\beta k}\hat{LS}_q^{(k)}(\mathbf{I})$ is bounded for any k, so $\hat{SS}_q(\mathbf{I})$ is still bounded.

3.5 Noise Calibration

After obtaining an $\hat{SS}_q(\mathbf{I})$, adding noise to the query answer to achieve differential privacy is straightforward. In particular, Nissim et al. [21] describe the following two mechanisms.

General Cauchy. The general Cauchy distribution has pdf $h(z) \propto \frac{1}{1+|z|^\gamma}$. It has bounded variance for $\gamma > 3$. It is shown [21] that by setting $\beta = \frac{\varepsilon}{2(\gamma+1)}$, adding noise $\frac{2(\gamma+1)\hat{SS}_q(\mathbf{I})}{\varepsilon}\eta$ to $|q(\mathbf{I})|$ preserves ε-differential privacy, where η is drawn the general Cauchy distribution. We use $\gamma = 4$, for which $\mathrm{Var}[\eta] = 1$, and the noise level (i.e., the standard deviation of the noise distribution) is thus $\frac{10}{\varepsilon}\hat{SS}_q(\mathbf{I})$.

Laplace. The general Cauchy distribution has a heavy tail. Alternatively, one can use the Laplace distribution with pdf $h(z) \propto e^{-|z|}$, which has a better concentration. However, adding noise according to the Laplace distribution only yields (ε, δ)-differential privacy. Specifically, one can set $\beta = \frac{\varepsilon}{2\ln(2/\delta)}$ and add noise $\frac{2\hat{SS}_q(\mathbf{I})}{\varepsilon}\eta$, where η is drawn from the Laplace distribution. Since $\mathrm{Var}[\eta] = 2$, the noise level is $\frac{2\sqrt{2}}{\varepsilon}\hat{SS}_q(\mathbf{I})$. Note that the noise level of using the Laplace distribution may not be smaller than that of general Cauchy, because for the same ε, the Laplace mechanism requires a smaller β, which in turn leads to a larger $\hat{SS}_q(\mathbf{I})$.

Note that the computation of $\hat{SS}_q(\mathbf{I})$ and the subsequent noise calibration step are both done internally; only the noise-masked query result will be published in the end.

4 RESIDUAL SENSITIVITY

4.1 Residual Queries and Boundaries

Given a multi-way join q, its *residual query* on a subset $E \subseteq [n]$ of relations is $q_E := \bowtie_{i\in E} R_i(\mathbf{x}_i)$. Its *boundary*, denoted ∂q_E, is the set of attributes that belong to atoms both in and out of E, i.e., $\partial q_E = \{x \mid x \in \mathbf{x}_i \cap \mathbf{x}_j, i \in E, j \in \bar{E}\}$. The following notion plays an important role in our development.

Definition 4.1 (Maximum boundary and witness). For a residual query q_E on database instance $\mathbf{I}$, its *maximum multiplicity over the boundary*, or simply *maximum boundary*, is defined as

$$T_E(\mathbf{I}) = \max_{t\in\mathbf{dom}(\partial q_E)} |q_E(\mathbf{I}) \ltimes t|.$$

A *witness tuple of the maximum multiplicity over the boundary*, or simply a *witness*, of q_E is

$$t_E(\mathbf{I}) = \arg\max_{t\in\mathbf{dom}(\partial q_E)} |q_E(\mathbf{I}) \ltimes t|. \tag{5}$$

Per convention, when $E = \emptyset$, q_E is an empty query and $q_E(\mathbf{I}) = \{\langle\rangle\}$, where $\langle\rangle$ denotes the empty tuple, thus $T_\emptyset(\mathbf{I}) = 1$.

Example 4.2. Figure 1 illustrates these concepts using the query $q = R_1(A,B,C) \bowtie R_2(D,E,F) \bowtie R_3(A,D) \bowtie R_4(C,F)$ on a specific database instance. The two residual queries shown are for $E = \{1,3\}$ and $E = \{1,2,3\}$. For the first residual query, the boundary, maximum boundary, and witness tuple are $\{C,D\}$, 2, and (c_1,d_1) respectively. For the second one, those are $\{C,F\}$, 4, and (c_1,f_1). □

Note that $T_E(\mathbf{I})$ can be computed by the following SQL query:

```
SELECT MAX(Boundary) FROM                                        (6)
    (SELECT COUNT(*) AS Boundary FROM q_E GROUP BY ∂q_E)
```

It is a special case of an *AJAR* query [12], and can be computed in $O(N^w)$ time, where w is a particular notion of the *width* of the query. The exact definition of w is a bit technical, but it is always a constant depending only on q_E and ∂q_E. Thus, $T_E(\mathbf{I})$ can be computed in polynomial time. Furthermore, various optimizations are possible; we discuss some of them in Section 6.2.

The following observations on the function $T_E(\cdot)$ are immediate. Recall that I_i is a shorthand for $\mathbf{I}(R_i)$.

Lemma 4.3. *For two instances $\mathbf{I}, \mathbf{I}'$ and any $E \subseteq [n]$, if $I_i = I_i'$ for all $i \in E$, then $T_E(\mathbf{I}) = T_E(\mathbf{I}')$.*

Lemma 4.4. *For $\mathbf{I}, \mathbf{I}'$, if $I_i \subseteq I_i'$ for all $i \in E$, then $T_E(\mathbf{I}) \le T_E(\mathbf{I}')$.*

Notation	Meaning
$\mathbf{I}, \mathbf{I}'$	Database instances
$I_1, \ldots, I_n$	Relation instances
q_E	Residual query
∂q_E	Boundary of q_E
T_E	Maximum boundary of q_E
t_E	Witness of q_E
$\mathcal{I}^k$	The set of instances having distance k to $\mathbf{I}$
$\mathbf{s}$	Distance vector used to describe the distance between two instances
$\mathbf{S}^k$	The set of distance vectors such that the total distance of all private relations is k
$\mathcal{I}^{\mathbf{s}}$	The set of instance differing from $\mathbf{I}$ with $\mathbf{s}$
$\hat{T}_{E,\mathbf{s}}(\mathbf{I})$	An upper bound of $T_E(\mathbf{I}')$ for any $\mathbf{I}' \in \mathcal{I}^{\mathbf{s}}$
GS_q	The global sensitivity of q
$LS_q(\mathbf{I})$	The local sensitivity of instance $\mathbf{I}$
$LS_q^{(k)}(\mathbf{I})$	The local sensitivity of at distance k from $\mathbf{I}$
$\hat{LS}_{q,\mathbf{s}}(\mathbf{I})$	An upper bound of $LS_q(\mathbf{I}')$ for any $\mathbf{I}' \in \mathcal{I}^{\mathbf{s}}$
$\hat{LS}_q^{(k)}(\mathbf{I})$	An upper bound of $LS_q^{(k)}(\mathbf{I}')$ for any $\mathbf{I}' \in \mathcal{I}^k$
$RS_q(\mathbf{I})$	Residual sensitivity of $\mathbf{I}$

Table 1: Notation used in the paper.

4.2 Formulating $LS_q^{(k)}$ with T_E

The residual sensitivity of a query is based on analyzing the sensitivity of (the maximum boundary of) its residual queries. First, we build a connection between $LS_q^{(k)}(\mathbf{I})$ and $T_E(\mathbf{I})$, starting with the simple case $LS_q^{(0)}(\mathbf{I}) = LS_q(\mathbf{I})$:

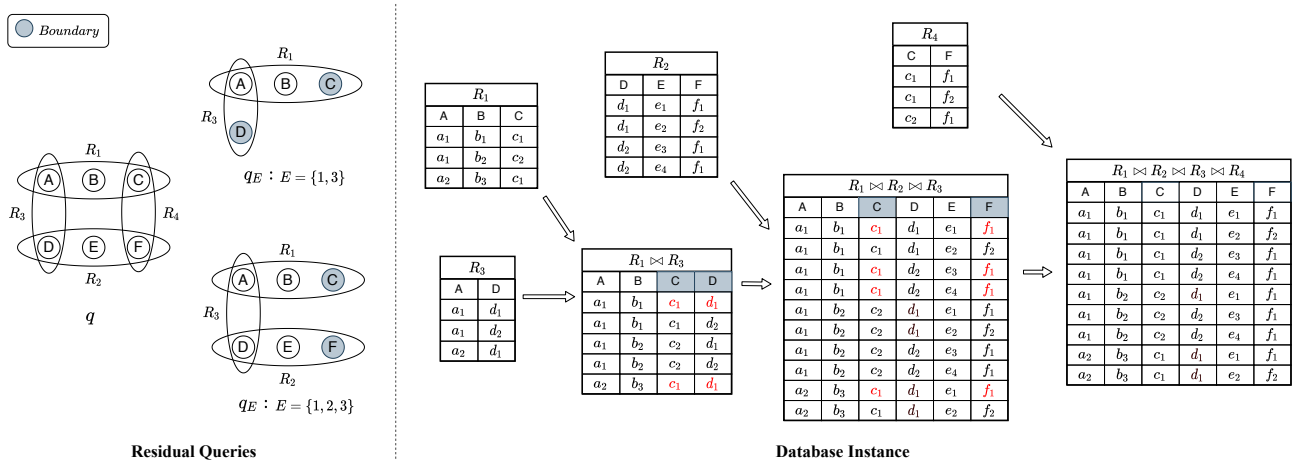

Figure 1: Residual query, boundary, maximum boundary, and witness.

Theorem 4.5. $LS_q(\mathbf{I}) = \max_{i \in P} T_{[n]-\{i\}}(\mathbf{I})$.

Proof. First, note that the definition of $LS_q(\mathbf{I})$ in (1) can be rewritten as

$$LS_q(\mathbf{I}) = \max_{i \in P} \max_{\mathbf{I}', d(\mathbf{I},\mathbf{I}')=1, d(I_i, I_i')=1} ||q(\mathbf{I})| - |q(\mathbf{I}')||.$$

Consider any $\mathbf{I}'$ with $d(\mathbf{I},\mathbf{I}') = 1, d(I_i, I_i') = 1$. I_i' can be different from I_i in three ways: insertion, deletion, or change of a tuple $t' \in \mathbf{dom}(\mathbf{x}_i)$. In the first case, $I_i' = I_i \cup \{t'\}, t' \notin I_i$, so

$$\begin{aligned} ||q(\mathbf{I})| - |q(\mathbf{I}')|| &= |q(\mathbf{I}')| - |q(\mathbf{I})| \\ &= |\bowtie_{j \in [n], j \neq i} (I_j \ltimes t')|. \end{aligned}$$

Similarly, for the second case, $I_i' = I_i - t', t' \in I_i$ and

$$||q(\mathbf{I})| - |q(\mathbf{I}')|| = |\bowtie_{j \in [n], j \neq i} (I_j \ltimes t')|.$$

Thus, over all $\mathbf{I}'$ that differs from $\mathbf{I}$ by the insertion or deletion of a tuple in I_i, we have

$$\begin{aligned} &\max_{\mathbf{I}'} ||q(\mathbf{I})| - |q(\mathbf{I}')|| \\ =&\max \left\{ \max_{t' \in \mathbf{I}} |\bowtie_{j \in [n], j \neq i} (I_j \ltimes t')|, \right. \\ &\left. \max_{t' \in \mathbf{dom}(\mathbf{x}_i), t' \notin \mathbf{I}} |\bowtie_{j \in [n], j \neq i} (I_j \ltimes t')| \right\} \\ =&\max_{t \in \mathbf{dom}(\partial q_{[n]-\{i\}})} |\bowtie_{j \in [n], j \neq i} (I_j \ltimes t)| \\ =&T_{[n]-\{i\}}(\mathbf{I}). \end{aligned}$$

For the third case where $\mathbf{I}'$ is obtained from $\mathbf{I}$ by changing a tuple in I_i, consider $\mathbf{I}'' = \mathbf{I} \cap \mathbf{I}'$. From previous work, we can derive

$$||q(\mathbf{I})| - |q(\mathbf{I}')|| \leq T_{[n]-\{i\}}(\mathbf{I}'').$$

Note that $\mathbf{I}$ and $\mathbf{I}''$ differ only in I_i, so $T_{[n]-\{i\}}(\mathbf{I}) = T_{[n]-\{i\}}(\mathbf{I}'')$ by Lemma 4.3. Thus, over all $\mathbf{I}'$ that differs from $\mathbf{I}$ by the changing one tuple, we have

$$\max_{\mathbf{I}'} ||q(\mathbf{I})| - |q(\mathbf{I}')|| \leq T_{[n]-\{i\}}(\mathbf{I}).$$

Summarizing the three cases, we have

$$\max_{\mathbf{I}', d(\mathbf{I},\mathbf{I}')=1, d(I_i, I_i')=1} ||q(\mathbf{I})| - |q(\mathbf{I}')|| = T_{[n]-\{i\}}(\mathbf{I}),$$

and

$$\begin{aligned} LS_q(\mathbf{I}) &= \max_{i \in P} \max_{\mathbf{I}', d(\mathbf{I},\mathbf{I}')=1, d(I_i, I_i')=1} ||q(\mathbf{I})| - |q(\mathbf{I}')|| \\ &= \max_{i \in P} T_{[n]-\{i\}}(\mathbf{I}). \end{aligned}$$

□

Next, consider $LS_q^{(k)}(\mathbf{I})$ for $k \geq 1$. Denote the set of instances having distance k to $\mathbf{I}$ as $\mathcal{I}^k = \{\mathbf{I}' : d(\mathbf{I},\mathbf{I}') = k\}$. By plugging Theorem 4.5 into (2), we have

$$LS_q^{(k)}(\mathbf{I}) = \max_{\mathbf{I}' \in \mathcal{I}^k} \max_{i \in P} T_{[n]-\{i\}}(\mathbf{I}'). \tag{7}$$

However, this formula does not yield a polynomial-time algorithm for computing $LS_q^{(k)}(\mathbf{I})$, as there are $O(N^k)$ instances in $\mathcal{I}^k$. Instead, we show how to efficiently compute an upper bound $\hat{LS}_q^{(k)}(\mathbf{I})$, which is needed in Theorem 3.4.

4.3 Sensitivity of T_E

Our upper bound $\hat{LS}_q^{(k)}(\mathbf{I})$ is based on analyzing the sensitivity of $T_E(\mathbf{I})$. The function $T_E(\mathbf{I})$ takes relation instances $I_i, i \in E$ as input and outputs a count, and its sensitivity is the maximum amount of change in $T_E(\mathbf{I})$ when $\mathbf{I}$ changes. Below, We first bound the sensitivity of $T_E(\mathbf{I})$ when only one tuple is changed. Then, we move onto the case where several tuples can change, but all the changes are in the same relation. Finally, we consider arbitrary changes.

Lemma 4.6. *Given any $E \subseteq [n], i \in E$, and two instances $\mathbf{I}, \mathbf{I}'$ such that $d(I_i, I_i') = 1, I_j = I_j'$ for all $j \in E - \{i\}$, we have $|T_E(\mathbf{I}) - T_E(\mathbf{I}')| \leq T_{E-\{i\}}(\mathbf{I})$.*

Proof. Similar to the proof of Theorem 4.5, there are also three cases: I_i' is obtained from I_i by the insertion, deletion or change of one tuple $t' \in \mathbf{dom}(\mathbf{x}_i)$.

For the first case, we have $T_E(\mathbf{I}') \geq T_E(\mathbf{I})$ by Lemma 4.4. Suppose the extra tuple in I_i' is t'.

Let $t_E(\mathbf{I}')$ be a witness of $T_E(\mathbf{I}')$ as defined in (5), and let

$$\tilde{T}_E(\mathbf{I}) = |q_E(\mathbf{I}) \ltimes t_E(\mathbf{I}')|.$$

By definition, $T_E(\mathbf{I}) \geq \tilde{T}_E(\mathbf{I})$, so

$$|T_E(\mathbf{I}) - T_E(\mathbf{I}')| = T_E(\mathbf{I}') - T_E(\mathbf{I}) \leq T_E(\mathbf{I}') - \tilde{T}_E(\mathbf{I}).$$

Now we bound $T_E(\mathbf{I}') - \tilde{T}_E(\mathbf{I})$:

$$\begin{aligned} &T_E(\mathbf{I}') - \tilde{T}_E(\mathbf{I}) \\ =&|q_E(\mathbf{I}') \ltimes t_E(\mathbf{I}')| - |q_E(\mathbf{I}) \ltimes t_E(\mathbf{I}')| \\ =&|(\ltimes_{j\in E-\{i\}} I_j) \bowtie t' \ltimes t_E(\mathbf{I}')| \\ =&|(\ltimes_{j\in E-\{i\}} I_j) \ltimes (t' \bowtie t_E(\mathbf{I}')|. \end{aligned} \tag{8}$$

Note that $\ltimes_{j\in E-\{i\}} I_j$ is just $q_{E-\{i\}}$ and $t' \bowtie t_E(\mathbf{I}')$ does not have any attribute interior to $q_{E-\{i\}}$. So (8) is at most $T_{E-\{i\}}(\mathbf{I})$ by definition.

The case where $\mathbf{I}$' has one less tuple than $\mathbf{I}$ is symmetric. Finally, the third case can be handled using a similar argument as in the proof of Theorem4.5. □

Lemma 4.7. *Given any $E \subseteq [n], i \in E$ and two instances $\mathbf{I}, \mathbf{I}'$ such that $d(I_i, I_i') = k, I_j = I_j'$ for all $j \in E-\{i\}$, we have $|T_E(\mathbf{I}) - T_E(\mathbf{I}')| \leq k \cdot T_{E-\{i\}}(\mathbf{I})$.*

Proof. For the given $\mathbf{I}, \mathbf{I}'$, there is a sequence of instances $\mathbf{I}^0, \mathbf{I}^1, \ldots, \mathbf{I}^k$, such that $\mathbf{I}^0 = \mathbf{I}, \mathbf{I}^k = \mathbf{I}'$, while any two neighboring instances differ by one tuple in I_i.

Based on Lemma 4.3 and Lemma 4.6, for every $\ell \in [k]$,

$$|T_E(\mathbf{I}^{\ell-1}) - T_E(\mathbf{I}^\ell)| \leq T_{E-\{i\}}(\mathbf{I}^{\ell-1}) = T_{E-\{i\}}(\mathbf{I}^0) = T_{E-\{i\}}(\mathbf{I}). \tag{9}$$

Summing (9) over all ℓ proves the lemma. □

Now, we consider the general case.

Lemma 4.8. *Given any $E \subseteq [n]$ and two instances $\mathbf{I}, \mathbf{I}'$, we have*

$$|T_E(\mathbf{I}) - T_E(\mathbf{I}')| \leq \sum_{E'\subseteq E, E'\neq\emptyset} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right).$$

Proof. For any $\mathbf{I}, \mathbf{I}'$, there is a sequence of instances $\mathbf{I}^0, \mathbf{I}^1, \ldots, \mathbf{I}^n$ such that $\mathbf{I}^0 = \mathbf{I}, \mathbf{I}^n = \mathbf{I}'$, while $\mathbf{I}^{\ell-1}$ and $\mathbf{I}^\ell$ differ only in I_ℓ, for $\ell \in [n]$. More precisely, $I_i^\ell = I_i'$ if $i \leq \ell$, and $I_i^\ell = I_i$ if $i \geq \ell + 1$.

We will prove by induction that, for every $\ell = 0, 1, \ldots, n$,

$$|T_E(\mathbf{I}^\ell) - T_E(\mathbf{I}^0)| \leq \sum_{E'\subseteq E\cap[\ell], E'\neq\emptyset} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right), \tag{10}$$

for any $E \subseteq [n]$.

For the base case $\ell = 0$, both sides of (10) are 0. For the inductive step, assume (10) holds on $\ell - 1$ for any E. We will prove that it also holds on ℓ for any E. For any given E, we divide the set $\{E' : E' \subseteq E \cap [\ell], E' \neq \emptyset\}$ into two subsets $\mathcal{E}_1 = \{E' : E' \subseteq E \cap [\ell], \ell \in E'\}$ and $\mathcal{E}_2 = \{E' : E' \subseteq E \cap [\ell-1], E' \neq \emptyset\}$, namely, $\mathcal{E}_1$ consists of E' that includes ℓ while $\mathcal{E}_2$ consists of those that do not. Consider the following two cases:

The easy case is when $\ell \notin E$. In this case, $\mathcal{E}_1$ is empty, and by Lemma 4.3, $T_E(\mathbf{I}^\ell) = T_E(\mathbf{I}^{\ell-1})$. Therefore,

$$\begin{aligned} &|T_E(\mathbf{I}^\ell) - T_E(\mathbf{I}^0)| \\ =&|T_E(\mathbf{I}^{\ell-1}) - T_E(\mathbf{I}^0)| \\ \leq& \sum_{E'\in\mathcal{E}_2} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right) \quad \text{(induction hypothesis)} \\ =& \sum_{E'\in\mathcal{E}_1\cup\mathcal{E}_2} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right), \end{aligned}$$

as desired.

The harder case is when $\ell \in E$. By Lemma 4.7, we have

$$|T_E(\mathbf{I}^\ell) - T_E(\mathbf{I}^{\ell-1})| \leq d(I_\ell, I_\ell') T_{E-\{\ell\}}(\mathbf{I}^{\ell-1}). \tag{11}$$

Define $\prod_{i\in E'} d(I_i, I_i') = 1$ if $E' = \emptyset$. From the induction hypothesis, we have

$$T_E(\mathbf{I}^{\ell-1}) \leq \sum_{E'\subseteq E\cap[\ell-1]} \left(T_{E-E'}(\mathbf{I}) \prod_{j\in E'} d(I_j, I_j') \right). \tag{12}$$

Recall that the induction hypothesis holds for any E. In particular, we use $E - \{\ell\}$ in place of E in (12), and plug it into (11):

$$\begin{aligned} &|T_E(\mathbf{I}^\ell) - T_E(\mathbf{I}^{\ell-1})| \\ \leq& d(I_\ell, I_\ell') \sum_{E'\subseteq(E-\{\ell\})\cap[\ell-1]} \left(T_{E-\{\ell\}-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right) \\ =& \sum_{E'\subseteq E\cap[\ell-1]} \left(T_{E-(E'\cup\{\ell\})}(\mathbf{I}) \prod_{i\in E'\cup\{\ell\}} d(I_i, I_i') \right) \\ =& \sum_{E'\subseteq E\cap[\ell], \ell\in E'} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right) \\ =& \sum_{E'\in\mathcal{E}_1} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right). \end{aligned} \tag{13}$$

Now we can finish the inductive step:

$$\begin{aligned} &|T_E(\mathbf{I}^\ell) - T_E(\mathbf{I}^0)| \\ \leq& |T_E(\mathbf{I}^\ell) - T_E(\mathbf{I}^{\ell-1})| + |T_E(\mathbf{I}^{\ell-1}) - T_E(\mathbf{I}^0)| \\ \leq& \sum_{E'\in\mathcal{E}_1} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right) \quad \text{(by (13))} \\ &+ \sum_{E'\in\mathcal{E}_2} \left(T_{E-E'}(\mathbf{I}) \prod_{i\in E'} d(I_i, I_i') \right) \quad \text{(induction hypothesis)} \\ =& \sum_{E'\in\mathcal{E}_1\cup\mathcal{E}_2} \left(T_{E-E'}(\mathbf{I}) \prod_{j\in E'} d(I_j, I_j') \right). \end{aligned}$$

□

4.4 $\hat{LS}_q^{(k)}$: An Upper Bound of $LS_q^{(k)}$

In order to use Lemma 4.8 to bound $LS_q^{(k)}$, we need to rewrite (7) at a finer granularity. First, for any two instances $\mathbf{I}, \mathbf{I}'$, their *distance vector* is $\mathbf{s} = (d(I_1, I_1'), \ldots, d(I_n, I_n'))$. For any $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{N}^n$,

let $\mathcal{I}^{\mathbf{s}} = \{\mathbf{I}' : d(I_i, I_i') = s_i, i \in [n]\}$. Let $\mathbf{S}^k$ be the set of distance vectors such that the total distance of all private relations is k, i.e.,

$$\mathbf{S}^k = \left\{ \mathbf{s} : \sum_{i \in P} s_i = k \text{ and } s_j = 0, j \in \bar{P} \right\}.$$

Note that $|\mathbf{S}^k| = O((n_P + k)^{n_P})$ and $\mathcal{I}^k = \cup_{\mathbf{s} \in \mathbf{S}^k} \mathcal{I}^{\mathbf{s}}$. Let

$$LS_{q,\mathbf{s}}(\mathbf{I}) = \max_{\mathbf{I}' \in \mathcal{I}^{\mathbf{s}}} \max_{i \in P} T_{[n]-\{i\}}(\mathbf{I}').$$

Then (7) can be rewritten as

$$LS_q^{(k)}(\mathbf{I}) = \max_{\mathbf{s} \in \mathbf{S}^k} LS_{q,\mathbf{s}}(\mathbf{I}).$$

Therefore, any upper bound of $LS_{q,\mathbf{s}}(\mathbf{I})$ yields an upper bound of $LS_q^{(k)}(\mathbf{I})$. Suppose $\hat{T}_{E,\mathbf{s}}(\mathbf{I})$ is an upper bound of $\max_{\mathbf{I}' \in \mathcal{I}^{\mathbf{s}}} T_E(\mathbf{I}')$. Then, we set our upper bound of $LS_{q,\mathbf{s}}(\mathbf{I})$ as

$$\hat{LS}_{q,\mathbf{s}}(\mathbf{I}) = \max_{i \in P} \hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I}), \tag{14}$$

and set the upper bound on $\hat{LS}_q^{(k)}(\mathbf{I})$ as

$$\hat{LS}_q^{(k)}(\mathbf{I}) = \max_{\mathbf{s} \in \mathbf{S}^k} \hat{LS}_{q,\mathbf{s}}(\mathbf{I}). \tag{15}$$

We compute $\hat{T}_{E,\mathbf{s}}(\mathbf{I})$ based on Lemma 4.8. For any $\mathbf{I}' \in \mathcal{I}^{\mathbf{s}}$, we have (recall that $\prod_{i \in E'} s_i$ is defined to be 1 if $E' = \emptyset$)

$$T_E(\mathbf{I}') \le T_E(\mathbf{I}) + |T_E(\mathbf{I}') - T_E(\mathbf{I})| \le \sum_{E' \subseteq E} \left(T_{E-E'}(\mathbf{I}) \prod_{i \in E'} s_i \right).$$

Thus,

$$\hat{T}_{E,\mathbf{s}}(\mathbf{I}) = \sum_{E' \subseteq E \cap P} \left(T_{E-E'}(\mathbf{I}) \prod_{i \in E'} s_i \right) \tag{16}$$

$$= \sum_{E' \subseteq E} \left(T_{E-E'}(\mathbf{I}) \prod_{i \in E'} s_i \right) \tag{17}$$

is a valid upper bound of $\max_{\mathbf{I}' \in \mathcal{I}^{\mathbf{s}}} T_E(\mathbf{I}')$. Note that (16) and (17) are equal since $s_i = 0$ for any $i \in \bar{P}$.

Example 4.9. Following the Example 4.2, we have $T_{\{1,3\}}(\mathbf{I}) = 2$, $T_{\{1,2,3\}}(\mathbf{I}) = 4$ and $T_{\{1,3,4\}}(\mathbf{I}) = 3$. Assume R_2 and R_4 are private. That is $P = \{2, 4\}$ and $n_P = 2$. For a given k,

$$\mathbf{S}^k = \{(0, 0, 0, k), (0, 1, 0, k-1), \ldots, (0, k, 0, 0)\}$$

and $|\mathbf{S}^k| = k + 1$. For $(0, k', 0, k - k') \in \mathbf{S}^k$, we have

$$\hat{T}_{\{1,2,3\},(0,k',0,k-k')} = T_{\{1,2,3\}} + T_{\{1,3\}} \cdot k' = 2k' + 4$$

$$\hat{T}_{\{1,3,4\},(0,k',0,k-k')} = T_{\{1,3,4\}} + T_{\{1,3\}} \cdot (k - k') = 2k - 2k' + 3$$

and

$$\hat{LS}^{(k)}(\mathbf{I}) = \max_{\mathbf{s} \in \mathbf{S}^k} \max(2k' + 4, 2k - 2k' + 3) = 2k + 4.$$

Unlike $LS_q^{(k)}$, $\hat{LS}_q^{(k)}$ can be computed in polynomial time. This is because the former requires us to consider a super-polynomially large space $|\mathcal{I}^k| = O(N^k)$, while the latter reduces the search space to $|\mathbf{S}^k| = O((n_P + k)^{n_P})$ by equation (15). Recall that n_P is the number of private relations, which is considered a constant. Each $\hat{LS}_{q,\mathbf{s}}(\mathbf{I})$ depends on n_P number of $\hat{T}_{E,\mathbf{s}}(\mathbf{I})$'s (equation (14)), while each $\hat{T}_{E,\mathbf{s}}(\mathbf{I})$ depends on $O(2^{n_P})$ number of $T_{E-E'}(\mathbf{I})$'s (equation (16)). Finally, recall that each $T_{E-E'}(\mathbf{I})$ can be computed in polynomial time.

In addition to computational efficiency, $\hat{LS}_q^{(k)}(\cdot)$ also satisfies condition (2) in Theorem 3.4. To prove this, we need the following technical lemma:

Lemma 4.10. *Given any $E \subseteq [n]$, any $i \in [n]$, two instances $\mathbf{I}, \mathbf{I}'$ such that $d(\mathbf{I}, \mathbf{I}') = 1$, $d(I_i, I_i') = 1$, and any two distance vectors $\mathbf{s} = (s_1, \ldots, s_n)$ and $\mathbf{s}' = (s_1', \ldots, s_n')$ such that*

$$\mathbf{s}' = (s_1, \ldots, s_{i-1}, s_i + 1, s_{i+1}, \ldots, s_n),$$

we have

$$\sum_{E' \subseteq E} \left(T_{E-E'}(\mathbf{I}) \prod_{j \in E'} s_j \right) \le \sum_{E' \subseteq E} \left(T_{E-E'}(\mathbf{I}') \prod_{j \in E'} s_j' \right).$$

Proof. If $i \notin E$, then for any $E' \subseteq E$, $j \in E - E'$, we have $I_j = I_j'$. Thus $T_{E-E'}(\mathbf{I}) = T_{E-E'}(\mathbf{I}')$ by Lemma 4.3. Meanwhile, $s_j = s_j'$ for all $j \in E'$. Therefore, $\sum_{E' \subseteq E} \left(T_{E-E'}(\mathbf{I}) \prod_{j \in E'} s_j \right) = \sum_{E' \subseteq E} \left(T_{E-E'}(\mathbf{I}') \prod_{j \in E'} s_j' \right)$ in this case.

If $i \in E$, we divide the set $\{E' \subseteq E\}$ into two subsets $\mathcal{E}_1 = \{E' : E' \subseteq E, i \in E'\}$ and $\mathcal{E}_2 = \{E' : E' \subseteq E, i \notin E'\}$. Note that there is one-to-one correspondence between the subsets in $\mathcal{E}_1$ and the subsets in $\mathcal{E}_2$, i.e., for any $E' \in \mathcal{E}_2$, $E' \cup \{i\} \in \mathcal{E}_1$, and vice versa.

For any $E' \in \mathcal{E}_1$, we have $i \notin E - E'$, so $T_{E-E'}(\mathbf{I}) = T_{E-E'}(\mathbf{I}')$ by Lemma 4.3. Thus,

$$\sum_{E' \in \mathcal{E}_1} \left(T_{E-E'}(\mathbf{I}) \prod_{j \in E'} s_j \right) = \sum_{E' \in \mathcal{E}_1} \left(T_{E-E'}(\mathbf{I}') \prod_{j \in E'} s_j \right). \tag{18}$$

For any $E' \in \mathcal{E}_2$, we have $i \in E - E'$. By Lemma 4.6, we have $T_{E-E'}(\mathbf{I}) \le T_{E-E'}(\mathbf{I}') + T_{E-E'-\{i\}}(\mathbf{I}')$. Therefore,

$$\begin{aligned}
&\sum_{E' \in \mathcal{E}_2} \left(T_{E-E'}(\mathbf{I}) \prod_{j \in E'} s_j \right) \\
&\le \sum_{E' \in \mathcal{E}_2} \left((T_{E-E'}(\mathbf{I}') + T_{E-E'-\{i\}}(\mathbf{I}')) \prod_{j \in E'} s_j \right) \\
&= \sum_{E' \in \mathcal{E}_2} \left(T_{E-E'}(\mathbf{I}') \prod_{j \in E'} s_j \right) + \sum_{E' \in \mathcal{E}_2} \left(T_{E-(E' \cup \{i\})}(\mathbf{I}') \prod_{j \in E'} s_j \right) \\
&= \sum_{E' \in \mathcal{E}_2} \left(T_{E-E'}(\mathbf{I}') \prod_{j \in E'} s_j \right) + \sum_{E' \in \mathcal{E}_1} \left(T_{E-E'}(\mathbf{I}') \prod_{j \in E'-\{i\}} s_j \right).
\end{aligned} \tag{19}$$

Note that the last step makes use of the one-to-one correspondence between $\mathcal{E}_1$ and $\mathcal{E}_2$.

Finally, based on (18) and (19), we have

$$\begin{aligned}
&\sum_{E'\subseteq E}\left(T_{E-E'}(\mathbf{I})\prod_{j\in E'}s_j\right)\\
&\le \sum_{E'\in\mathcal{E}_1}\left(T_{E-E'}(\mathbf{I}')(\prod_{j\in E'}s_j+\prod_{j\in E'-\{i\}}s_j)\right)+\sum_{E'\in\mathcal{E}_2}\left(T_{E-E'}(\mathbf{I}')\prod_{j\in E'}s_j\right)\\
&= \sum_{E'\in\mathcal{E}_1}\left(T_{E-E'}(\mathbf{I}')(s_i+1)\prod_{j\in E'-\{i\}}s_j\right)+\sum_{E'\in\mathcal{E}_2}\left(T_{E-E'}(\mathbf{I}')\prod_{j\in E'}s_j\right)\\
&= \sum_{E'\in\mathcal{E}_1}\left(T_{E-E'}(\mathbf{I}')\prod_{j\in E'}s'_j\right)+\sum_{E'\in\mathcal{E}_2}\left(T_{E-E'}(\mathbf{I}')\prod_{j\in E'}s'_j\right)\\
&= \sum_{E'\subseteq E}\left(T_{E-E'}(\mathbf{I}')\prod_{j\in E'}s'_j\right),
\end{aligned}$$

□

With Lemma 4.10, we can show the smoothness property of $\hat{LS}^{(k)}(\cdot)$.

Lemma 4.11. *For any $\mathbf{I},\mathbf{I}'$ such that $d(\mathbf{I},\mathbf{I}')=1$, $\hat{LS}_q^{(k)}(\mathbf{I})\le\hat{LS}_q^{(k+1)}(\mathbf{I}')$ for any $k\ge 0$.*

Proof. By definition (15), we have

$$\hat{LS}_q^{(k+1)}(\mathbf{I})=\max_{\mathbf{s}\in\mathbf{S}^{k+1}}\hat{LS}_{q,\mathbf{s}}(\mathbf{I}).$$

So it suffices to find one $\mathbf{s}^{\mathbf{I}'}\in\mathbf{S}^{k+1}$ such that

$$\hat{LS}_q^{(k)}(\mathbf{I})\le\hat{LS}_{q,\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}').$$

Assume $\mathbf{I},\mathbf{I}'$ differ by one tuple in $R_i, i\in P$. Let

$$\mathbf{s}^{\mathbf{I}}=(s_1^{\mathbf{I}},\dots,s_n^{\mathbf{I}})=\arg\max_{\mathbf{s}\in\mathbf{S}^k}\hat{LS}_{q,\mathbf{s}}(\mathbf{I}).$$

Below we show that $\mathbf{s}^{\mathbf{I}'}=(s_1^{\mathbf{I}},\dots s_{i-1}^{\mathbf{I}},s_i^{\mathbf{I}}+1,s_{i+1}^{\mathbf{I}},\dots,s_n^{\mathbf{I}})\in\mathbf{S}^{k+1}$ meets the requirement.

From (14) and (15), we have

$$\begin{aligned}
\hat{LS}_q^{(k)}(\mathbf{I})&=\hat{LS}_{q,\mathbf{s}^{\mathbf{I}}}(\mathbf{I})=\max_{j\in P}\hat{T}_{[n]-\{j\},\mathbf{s}^{\mathbf{I}}}(\mathbf{I}),\\
\hat{LS}_q^{(k+1)}(\mathbf{I}')&\ge\hat{LS}_{q,\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}')=\max_{j\in P}\hat{T}_{[n]-\{j\},\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}').
\end{aligned}$$

Thus, it is sufficient to show that $\hat{T}_{[n]-\{j\},\mathbf{s}^{\mathbf{I}}}(\mathbf{I})\le\hat{T}_{[n]-\{j\},\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}')$ for any $j\in P$. In fact, we prove a stronger result that for any $E\subseteq[n]$, $\hat{T}_{E,\mathbf{s}^{\mathbf{I}}}(\mathbf{I})\le\hat{T}_{E,\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}')$.

We expand $\hat{T}_{E,\mathbf{s}^{\mathbf{I}}}(\mathbf{I})$ and $\hat{T}_{E,\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}')$ with (17):

$$\begin{aligned}
\hat{T}_{E,\mathbf{s}^{\mathbf{I}}}(\mathbf{I})&=\sum_{E'\subseteq E}\left(T_{E-E'}(\mathbf{I})\prod_{j\in E'}s_j^{\mathbf{I}}\right),\\
\hat{T}_{E,\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}')&=\sum_{E'\subseteq E}\left(T_{E-E'}(\mathbf{I}')\prod_{j\in E'}s_j^{\mathbf{I}'}\right).
\end{aligned}$$

Based on Lemma 4.10, we have for any $E\subseteq[n]$,

$$\sum_{E'\subseteq E}\left(T_{E-E'}(\mathbf{I})\prod_{j\in E'}s_j^{\mathbf{I}}\right)\le\sum_{E'\subseteq E}\left(T_{E-E'}(\mathbf{I}')\prod_{j\in E'}s_j^{\mathbf{I}'}\right),$$

so $\hat{T}_{E,\mathbf{s}^{\mathbf{I}}}(\mathbf{I})\le\hat{T}_{E,\mathbf{s}^{\mathbf{I}'}}(\mathbf{I}')$, as claimed. □

4.5 Residual Sensitivity

Based on the development so far, we can define and compute the *residual sensitivity* of a multi-way join query q on instance $\mathbf{I}$, denoted $RS_q(\mathbf{I})$, straightforwardly. We simply invoke Theorem 3.4 with the $\hat{LS}_q^{(k)}(\mathbf{I})$ as described in the previous section. More precisely, we take the following two simple steps to compute $RS_q(\mathbf{I})$:

(1) Compute $T_{\tilde{E}}(\mathbf{I})$ for every $E\subseteq P, E\ne\emptyset$.
(2) Compute

$$\begin{aligned}
RS_q(\mathbf{I})&=\max_{k\ge0}\left(e^{-\beta k}\min(\hat{GS}_q,\hat{LS}_q^{(k)}(\mathbf{I}))\right)\\
&=\max_{k\ge0}\left(e^{-\beta k}\min(\hat{GS}_q,\max_{\mathbf{s}\in\mathbf{S}^k}\max_{i\in P}\hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I}))\right),
\end{aligned}\tag{20}$$

where $\hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I})$ is calculated as in (16).

In order to compute (20), we first need to argue that we only have to consider a finite number of k's. This is not obvious as $\hat{GS}_q=\infty$ in our case.

Lemma 4.12. *The $\max_k$ in (20) achieves maximum on some $k\le\hat{k}=\frac{n_P-1}{1-e^{-\beta}}$.*

Proof. We first rewrite (20) as

$$RS_q(\mathbf{I})=\max_{i\in P}\max_{0\le k}\max_{\mathbf{s}\in\mathbf{S}^k}\left(e^{-\beta k}\min(\hat{GS}_q,\hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I}))\right).$$

First, for fixed i,k, $\max_{\mathbf{s}\in\mathbf{S}^k}\left(e^{-\beta k}\min(\hat{GS}_q,\hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I}))\right)$ only depends on $\mathbf{s}\in\mathbf{S}^k$ with $s_i=0$, since s_i does not affect the value of $\hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I})$. Now, consider any $k\ge\frac{n_P-1}{1-e^{-\beta}}$. We will show that for any $\mathbf{s}\in\mathbf{S}^k$ with $s_i=0$, there is an $\mathbf{s}'\in\mathbf{S}^{k-1}$ such that

$$e^{-\beta k}\hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I})\le e^{-\beta(k-1)}\hat{T}_{[n]-\{i\},\mathbf{s}'}(\mathbf{I}),\tag{21}$$

which means that such a k cannot yield the maximum.

Since $k\ge\frac{n_P-1}{1-e^{-\beta}}$, $\mathbf{s}\in\mathbf{S}^k, s_i=0$, there must exist one $j'\in[n]$ such that $s_{j'}\ge\frac{1}{1-e^{-\beta}}$. Then, we define $\mathbf{s}'$ as

$$s'_j=\begin{cases}s_j-1, & j=j';\\ s_j, & j\ne j'.\end{cases}$$

By (16), we have

$$\hat{T}_{[n]-\{i\},\mathbf{s}}(\mathbf{I})=\sum_{E'\subseteq P-\{i\}}\left(T_{[n]-E'-\{i\}}(\mathbf{I})\prod_{j\in E'}s_j\right).$$

We will show that

$$e^{-\beta k}T_{[n]-E'-\{i\}}(\mathbf{I})\prod_{j\in E'}s_j\le e^{-\beta(k-1)}T_{[n]-E'-\{i\}}(\mathbf{I})\prod_{j\in E'}s'_j\tag{22}$$

for any $E'\subseteq P-\{i\}$, thereby proving (21).

Note that (22) can be simplified to

$$\prod_{j\in E'}s_j\le e^{\beta}\prod_{j\in E'}s'_j.\tag{23}$$

If $j' \notin E'$, (23) trivially holds since $s_j = s'_j$ for all $j \in E'$ and $e^\beta \geq 1$. For the case $j' \in E'$, (23) simplifies to

$$s_{j'} \leq e^\beta (s_{j'} - 1),$$

which is equivalent to $s_{j'} \geq \frac{1}{1-e^{-\beta}}$. □

Example 4.13. Following Example 4.9, set $\beta = 0.1, \hat{GS}_q = \infty$. Since $\hat{k} = \frac{n_P - 1}{1-e^{-\beta}} < 11$, we calculate $\hat{LS}_q^{(k)}(\mathbf{I}) = 2k+4$ for $0 \leq k \leq 10$. Finally, we compute

$$RS_q(\mathbf{I}) = \max_{0 \leq k \leq 10} \left(e^{-0.1k} \hat{LS}_q^{(k)}(\mathbf{I}) \right) \approx 8.987,$$

where the maximum is attained at $k = 8$.

As mentioned in Section 4.1, step (1) takes polynomial time. Step (2) takes time $O(\hat{k} \cdot \hat{k}^{n_P} \cdot n_P) = O(1)$. Thus, we conclude:

Theorem 4.14. *Given any multi-way join query q, a set of private relations P, and an instance $\mathbf{I}$, its residual sensitivity $RS_q(\mathbf{I})$ can be computed in time polynomial in $N = |\mathbf{I}|$.*

4.6 Extensions

Selections and projections. So far we have only considered joins while ignored selection conditions. There are two types of selection conditions: (1) Selection conditions on individual relations, such as `Orders.Orderdate > 2020-01-01` in the example query in Section 1, can be applied in a preprocessing stage. (2) For selection conditions involving attributes in different relations, e.g., `Lineitem.Shipdate > Orders.Orderdate + 10 days`, we can ignore them when computing the residual sensitivity. It will still be a valid smooth upper bound of local sensitivity, since the presence of these selection conditions can only make the local sensitivity smaller. However, we can no longer guarantee that it is a constant-factor approximation of smooth sensitivity. The same holds for projection, since it does not increase sensitivity, either.

Group-by. A query with group-by returns a vectored output. It is known [21] that one can add noise to each component of the vector according to a smooth upper bound on local sensitivity of the ℓ_1-norm of the vector. Thus, for a query with a group-by clause, we first compute its residential sensitivity ignoring the group-by clause, and then add noise calibrated with the residential sensitivity to each group's count. A similar idea has been used in [13, 17].

5 ANALYTICAL COMPARISONS

In this section, we give analytical comparisons between residual sensitivity and previous notions of sensitivity for multi-way joins. An empirical comparison will be conducted in Section 6.

5.1 Compare with Smooth Sensitivity

Recall that the *smooth sensitivity* $SS_q(\mathbf{I})$ is the optimum, in the sense that it is the smallest smooth upper bound of local sensitivity. However, it takes super-polynomial time to compute $SS_q(\mathbf{I})$. In this section, we show that the residual sensitivity $RS_q(\mathbf{I})$, which is polynomial-time computable, is at most a constant factor larger than $SS_q(\mathbf{I})$, for any multi-way join q on any instance $\mathbf{I}$.

Recall that $SS_q(\mathbf{I})$ is defined in terms of $LS_q^{(k)}$ (see equation (3)), while $RS_q(\mathbf{I})$ depends on $T_{\bar{E}}(\mathbf{I})$, the maximum boundary of residual queries. We first build a connection between the two.

Lemma 5.1. *For any $E \subseteq P, E \neq \emptyset, LS_q^{(n_P-1)}(\mathbf{I}) \geq T_{\bar{E}}(\mathbf{I})$.*

Proof. We will construct an $\mathbf{I}'$ from $\mathbf{I}$ such that $d(\mathbf{I}, \mathbf{I}') \leq n_P - 1$ and $LS_q(\mathbf{I}') \geq T_{\bar{E}}(\mathbf{I})$. Let $t_{\bar{E}}(\mathbf{I})$ be a witness of $q_{\bar{E}}(\mathbf{I})$. Then, $T_{\bar{E}}(\mathbf{I})$ can be written as

$$T_{\bar{E}}(\mathbf{I}) = | \ltimes_{i \in \bar{E}} (I_i \ltimes t_{\bar{E}}(\mathbf{I}))|.$$

Recall that the attributes of $t_{\bar{E}}(\mathbf{I})$ are $\partial q_{\bar{E}}$. Note that at least one relation in E must have an attribute in $\partial q_{\bar{E}}$ (otherwise q would be disconnected). We may thus assume without loss of generality that $P = [n_P], E = [|E|]$.

We construct $\mathbf{I}'$ as follows. First, fix a tuple $t' \in \mathbf{dom}(\cup_{i \in E} \mathbf{x}_i)$ such that $\pi_{\partial q_{\bar{E}}} t' = t_{\bar{E}}(\mathbf{I})$, namely, t' is consistent with $t_{\bar{E}}(\mathbf{I})$ on all its attributes. For other attributes, their values in t' can be arbitrary. Then, for each $i \in [2, |E|]$, we add the tuple $\pi_{\mathbf{x}_i} t'$ to I_i, unless it already exists in I_i. It is clear that we add at most $|E| - 1 \leq n_P - 1$ tuples to $\mathbf{I}$, so $d(\mathbf{I}, \mathbf{I}') \leq n_P - 1$, and $LS_q^{(n_P-1)}(\mathbf{I}) \geq LS_q(\mathbf{I}')$.

Thus, it suffices to show $LS_q(\mathbf{I}') \geq T_{\bar{E}}(\mathbf{I})$. In doing so, we show that by flipping the tuple $t_1 = \pi_{\mathbf{x}_1} t'$ in I'_1, $|q(\mathbf{I}')|$ will change by at least $T_{\bar{E}}(\mathbf{I})$. Suppose $t_1 \notin I'_1$. Then by adding t_1 to I'_1, all query results involving t_1 will be added to $q(\mathbf{I}')$. The number of such tuples is

$$\begin{aligned} &|(\bowtie_{i \in [2,n]} I'_i) \bowtie t_1| \\ \geq &|(\bowtie_{i \in \bar{E}} I_i) \bowtie (\bowtie_{i \in E} \pi_{\mathbf{x}_i} t')| \\ = &|(\bowtie_{i \in \bar{E}} I_i) \bowtie t'| \\ = &|(\bowtie_{i \in \bar{E}} I_i) \ltimes t_{\bar{E}}(\mathbf{I})| = T_{\bar{E}}(\mathbf{I}), \end{aligned}$$

where the first inequality is due to the fact that every $I'_i, i \in E - \{1\}$ has at least one tuple inconsistent with t'. For the case $t_1 \in I'_1$, we remove t_1 from I'_1, and $|q(\mathbf{I}')|$ will decrease by the same amount. □

Next, we compare $\hat{LS}_q^{(k)}(\mathbf{I})$ with $LS_q^{(n_P-1)}(\mathbf{I})$.

Lemma 5.2. $\hat{LS}_q^{(k)}(\mathbf{I}) \leq (2k)^{n_P-1} LS_q^{(n_P-1)}(\mathbf{I})$.[3]

Proof. From (14), (15), (16), we have

$$\hat{LS}_q^{(k)}(\mathbf{I}) = \max_{\mathbf{s} \in \mathbb{S}^k} \max_{i \in P} \sum_{E' \subseteq P - \{i\}} \left(T_{[n]-E'-\{i\}}(\mathbf{I}) \prod_{j \in E'} s_j \right). \tag{24}$$

Since $E' \cup \{i\} \subseteq P, E' \cup \{i\} \neq \emptyset$, we can apply Lemma 5.1:

$$T_{[n]-E'-\{i\}}(\mathbf{I}) \leq LS_q^{(n_P-1)}(\mathbf{I}). \tag{25}$$

For any $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{S}^k$, $j \in P$, we have $s_j \leq k$. Thus,

$$\prod_{j \in E'} s_j \leq k^{|E'|} \leq k^{n_P - 1}. \tag{26}$$

[3] Define $0^0 = 1$.

Plugging (25), (26) into (24), we have

$$\begin{aligned}\hat{LS}_q^{(k)}(\mathbf{I}) &= \max_{\mathbf{s}\in\mathbf{S}^k}\max_{i\in P}\sum_{E'\subseteq P-\{i\}}\left(T_{[n]-E'-\{i\}}(\mathbf{I})\prod_{j\in E'}s_j\right)\\ &\le \max_{\mathbf{s}\in\mathbf{S}^k}\max_{i\in P}\sum_{E'\subseteq P-\{i\}}\left(LS_q^{(n_P-1)}(\mathbf{I})\cdot k^{n_P-1}\right)\\ &= (2k)^{n_P-1}LS_q^{(n_P-1)}(\mathbf{I}).\end{aligned}$$

□

Remark. When $n_P = 1$, Lemma 5.2 suggests that $\hat{LS}_q^{(k)}(\mathbf{I}) \le LS_q^{(0)}(\mathbf{I}) = LS_q(\mathbf{I})$. Meanwhile, because $\hat{LS}_q^{(k)}(\mathbf{I}) \ge LS_q^{(k)}(\mathbf{I}) \ge LS_q(\mathbf{I})$ by definition, it must be the case that $\hat{LS}_q^{(k)}(\mathbf{I}) = LS_q^{(k)}(\mathbf{I}) = LS_q(\mathbf{I})$. In fact, the second equality also follows from Theorem 4.5, which states that $LS_q(\mathbf{I}) = T_{[n]-\{i\}}(\mathbf{I})$, when P has only one relation i. Since $T_{[n]-\{i\}}(\mathbf{I})$ does not depend on I_i, $LS_q(\mathbf{I}')$ is the same for all $\mathbf{I}'$ that differs from $\mathbf{I}$ only in I_i, no matter how many tuples are different. Furthermore, when P has only one relation i, our upper bound $\hat{LS}_q^{(k)}(\mathbf{I}) = T_{[n]-\{i\}}(\mathbf{I})$ is the same as $LS_q(\mathbf{I})$, therefore has no approximation error.

Finally, we are ready to relate $RS_q(\mathbf{I})$ with $SS_q(\mathbf{I})$:

Theorem 5.3. *For any multi-way join query q on any instance $\mathbf{I}$ with n_P private relations, $RS_q(\mathbf{I}) \le \left(\frac{2(n_P-1)}{\beta e^{1-\beta}}\right)^{n_P-1}\cdot SS_q(\mathbf{I})$.*

Proof. Recall the definition of $SS_q(\mathbf{I})$ and $RS_q(\mathbf{I})$ in (3) and (4):

$$\begin{aligned}SS_q(\mathbf{I}) &= \max_{k\ge 0} e^{-\beta k}LS_q^{(k)}(\mathbf{I}),\\ RS_q(\mathbf{I}) &= \max_{0\le k\le\hat{k}}\left(e^{-\beta k}\min(\hat{LS}_q^{(k)}(\mathbf{I}), \hat{GS}_q)\right).\end{aligned}$$

Let

$$k_{RS} = \arg\max_{0\le k\le\hat{k}}\left(e^{-\beta k}\min(\hat{LS}_q^{(k)}(\mathbf{I}), \hat{GS}_q)\right).$$

Define the function

$$g(k) = e^{-\beta k}(2k)^{n_P-1}LS_q^{(n_P-1)}(\mathbf{I}).$$

Setting its derivative to 0, we see that $g(k)$ maximizes at $k_{\max} = \frac{n_P-1}{\beta}$ (even allowing k to take fractional values), so

$$g(k) \le g\left(\frac{n_P-1}{\beta}\right), \tag{27}$$

for all k. Therefore,

$$\begin{aligned}RS_q(\mathbf{I}) &\le e^{-\beta k_{RS}}\hat{LS}_q^{(k_{RS})}(\mathbf{I})\\ &\le e^{-\beta k_{RS}}(2k_{RS})^{n_P-1}(\mathbf{I})LS_q^{(n_P-1)}(\mathbf{I})\\ &\le e^{-(n_P-1)}\left(\frac{2(n_P-1)}{\beta}\right)^{n_P-1}LS_q^{(n_P-1)}(\mathbf{I})\\ &\le \left(\frac{2(n_P-1)}{\beta e^{1-\beta}}\right)^{n_P-1}\max_{k\ge 0}e^{-\beta k}LS_q^{(k)}(\mathbf{I})\\ &= \left(\frac{2(n_P-1)}{\beta e^{1-\beta}}\right)^{n_P-1}SS_q(\mathbf{I}),\end{aligned}$$

where the second inequality is by Lemma 5.2, and the third inequality is due to (27). □

Remark. When $n_P = 1$, Theorem 5.3 suggests that $RS_q(\mathbf{I}) = SS_q(\mathbf{I})$. Indeed, as discussed in the remark following Lemma 5.2, $LS_q^{(k)}(\mathbf{I}) = LS_q(\mathbf{I})$ for all k when $n_P = 1$, thus $SS_q(\mathbf{I}) = LS_q(\mathbf{I})$. Meanwhile, if there is only one private relation R_i, $LS_q(\mathbf{I}) = LS_q(I_i; \mathbf{I}-I_i)$ does not actually depend on the private data I_i, so it can be essentially considered as the global sensitivity of q over the public relation instances $\mathbf{I}-I_i$. Therefore, all the sensitivity measures, GS_q, LS_q, $LS_q^{(k)}$, SS_q, RS_q collapse into one in this case.

5.2 Compare with Elastic Sensitivity

In this section, we compare residual sensitivity $RS_q(\mathbf{I})$ with elastic sensitivity $ES_q(\mathbf{I})$, both of which are smooth upper bounds of local sensitivity, hence can be used directly for perturbing sensitive query results. Both can be computed in polynomial time; elastic sensitivity can be even computed in linear time $O(N)$. However, in this section we show that $RS_q(\mathbf{I}) \le ES_q(\mathbf{I})$ for any multi-way join q and instance $\mathbf{I}$, while there are cases where $ES_q(\mathbf{I})$ is much larger than $RS_q(\mathbf{I})$.

Elastic sensitivity [13] is defined in a similar form as smooth sensitivity and residual sensitivity:

$$ES_q(\mathbf{I}) = \max_{0\le k\le N}\left(e^{-\beta k}\tilde{LS}_q^{(k)}(\mathbf{I})\right),$$

where $\tilde{LS}_q^{(k)}(\mathbf{I})$ is also an upper bound of $LS_q^{(k)}(\mathbf{I})$. Thus, to show $RS_q(\mathbf{I}) \le ES_q(\mathbf{I})$, it suffices to show that $\forall k, \hat{LS}_q^{(k)}(\mathbf{I}) \le \tilde{LS}_q^{(k)}(\mathbf{I})$.

Acyclic queries. We first describe how $\tilde{LS}_q^{(k)}(\mathbf{I})$ is computed for an *acyclic query* q. In an acyclic query, its relations can be organized as a *join tree* $\mathcal{T}$, such that for each attribute A, all relations containing A are connected in $\mathcal{T}$. For convenience, we do not distinguish relation and node in join tree below. By specifying a relation R_i as the root, $\mathcal{T}$ can be made into a rooted tree, and we use $\mathcal{T}^i$ to denote the join tree rooted by R_i. For R_j, $j \ne i$, let $R_{p(j,i)}$ be its parent in $\mathcal{T}^i$. For $j \in [n]$, $\mathbf{x} \subseteq \mathbf{x}_j$, let $mf(\mathbf{x}, I_j)$ be the *maximum frequency* in I_j on attributes $\mathbf{x}$, i.e., $mf(\mathbf{x}, I_j) = \max_{t\in\mathbf{dom}(\mathbf{x})}|I_j \ltimes t|$.

Given an acyclic query q and its join tree $\mathcal{T}$, elastic sensitivity computes $\tilde{LS}_q^{(k)}(\mathbf{I})$ as

$$\begin{aligned}\tilde{LS}_q^{(k)}(\mathbf{I}) = \max_{i\in P}\Bigg(&\prod_{j\in P-\{i\}}(mf(\mathbf{x}_j\cap\mathbf{x}_{p(j,i)}, I_j)+k)\\ &\cdot\prod_{j\in[n]-P}mf(\mathbf{x}_j\cap\mathbf{x}_{p(j,i)}, I_j)\Bigg).\end{aligned} \tag{28}$$

Lemma 5.4. *For any acyclic query q, $RS_q(\mathbf{I}) \le ES_q(\mathbf{I})$.*

Proof. We first define some terminology. Recall the definition of the residual query $q_E = \bowtie_{j\in E}R_j(\mathbf{x}_j)$. Its boundary is ∂q_E, consists of variables shared by relations both in and out of E. Given a rooted join tree $\mathcal{T}^i, i \in [n]$, the *residual join tree* $\mathcal{T}_E^i$, $i \notin E$, consists of the relations $R_j, j \in E$ and the edges between them in $\mathcal{T}^i$. $\mathcal{T}_E^i$ may contain several connected components. We use $\mathcal{C}_E^i$ to denote the set of connected components of $\mathcal{T}_E^i$. For each connected component $C \in \mathcal{C}_E^i$, let its root in $\mathcal{T}_E^i$ be $R_{r(C)}$. We call these roots $R_{r(C)}, C \in \mathcal{C}_E^i$ the roots of $\mathcal{T}_E^i$. The *top boundary* of $\mathcal{T}_E^i$, denoted by $\partial\mathcal{T}_E^i$, is a set

of variables shared by the roots of $\mathcal{T}_E^i$ with their parents. It is easy to see that $\partial\mathcal{T}_E^i \subseteq \partial q_E$ since the parent of any root is not in E.

Since $\tilde{LS}_q^{(k)}(\mathbf{I})$ is defined in terms of maximal frequency while $\hat{LS}_q^{(k)}(\mathbf{I})$ depends on maximal boundary, we first build a connection between the two. For any rooted join tree $\mathcal{T}^i, i \in [n]$, and any $E \subseteq [n], i \notin E$, we have

$$\begin{aligned}
&\prod_{j\in E} mf(\mathbf{x}_j \cap \mathbf{x}_{p(j,i)}, I_j) \\
\geq &\max_{t\in\mathbf{dom}(\partial\mathcal{T}_E^i)} |(\ltimes_{j\in E} I_j) \ltimes t| \\
\geq &\max_{t\in\mathbf{dom}(\partial q_E)} |(\ltimes_{j\in E} I_j) \ltimes t| = T_E(\mathbf{I}).
\end{aligned}$$

The second inequality follows from $\partial\mathcal{T}_E^i \subseteq \partial q_E$, while the first inequality holds because

$$\begin{aligned}
&\max_{t\in\mathbf{dom}(\partial\mathcal{T}_E^i)} |(\ltimes_{j\in E} I_j) \ltimes t| \\
\leq &\prod_{C\in C_E^i} \max_{t\in\mathbf{dom}(\mathbf{x}_{r(C)}\cap\mathbf{x}_{p(r(C),i)})} |(\ltimes_{j\in C} I_j) \ltimes t| \\
= &\prod_{C\in C_E^i} \max_{t\in\mathbf{dom}(\mathbf{x}_{r(C)}\cap\mathbf{x}_{p(r(C),i)})} |(\ltimes_{j\in C-\{r(C)\}} I_j) \ltimes (I_{r(C)} \ltimes t)| \\
\leq &\prod_{j\in E} mf(\mathbf{x}_j \cap \mathbf{x}_{p(j,i)}, I_j). \qquad (29)
\end{aligned}$$

The derivation of (29) is similar to Lemma 1 in [13].

Therefore, we have

$$\prod_{j\in E} mf(\mathbf{x}_j \cap \mathbf{x}_{p(j,i)}, I_j) \geq T_E(\mathbf{I}). \qquad (30)$$

We can now compare $\hat{LS}_q^{(k)}(\mathbf{I})$ and $\tilde{LS}_q^{(k)}(\mathbf{I})$. Recall (24) and (28). To prove the lemma, it is sufficient to show that, for any i and $\mathbf{s} \in \mathbf{S}^k$,

$$\begin{aligned}
&\prod_{j\in P-\{i\}} \left(mf(\mathbf{x}_j \cap \mathbf{x}_{p(j,i)}, I_j) + k\right) \prod_{j\in[n]-P} mf\left(\mathbf{x}_j \cap \mathbf{x}_{p(j,i)}, I_j\right) \\
\geq &\sum_{E'\subseteq P-\{i\}} \left(T_{[n]-E'-\{i\}}(\mathbf{I}) \prod_{j\in E'} s_j\right). \qquad (31)
\end{aligned}$$

Expand the LHS of (31):

$$\text{LHS of (31)} = \sum_{E'\subseteq P-\{i\}} \left(k^{|E'|} \prod_{j\in[n]-E'-\{i\}} mf(\mathbf{x}_j \cap \mathbf{x}_{p(j,i)}, I_j)\right).$$

By (30), we have

$$T_{[n]-E'-\{i\}}(\mathbf{I}) \leq \prod_{j\in[n]-E'-\{i\}} mf(\mathbf{x}_j \cap \mathbf{x}_{p(j,i)}, I_j).$$

Meanwhile, it is clear that $\prod_{j\in E'} s_j \leq k^{|E'|}$, (31) is thus proved. □

Cyclic queries. To compute the elastic sensitivity for a cyclic query q, Johnson et al. [13] simply remove some join conditions to form an acyclic query q', and set $ES_q(\mathbf{I}) = ES_{q'}(\mathbf{I})$. On the other hand, we have $RS_q(\mathbf{I}) \leq RS_{q'}(\mathbf{I})$, since the residual sensitivity is defined in terms of maximal boundary of residual queries, which cannot decrease after removing some join conditions. Thus,

$$RS_q(\mathbf{I}) \leq RS_{q'}(\mathbf{I}) \leq ES_{q'}(\mathbf{I}) = ES_q(\mathbf{I}).$$

So we conclude:

Theorem 5.5. *For any multi-way join query q and any instance $\mathbf{I}$, $RS_q(\mathbf{I}) \leq ES_q(\mathbf{I})$.*

The gap. We have proved that $RS_q(\mathbf{I}) \leq ES_q(\mathbf{I})$ for all multi-way join queries and instances. Below, we show that on certain queries and instances, the former is $O(1)$ while the latter is $\Theta(N^{n-1})$, which is as large as the global sensitivity.

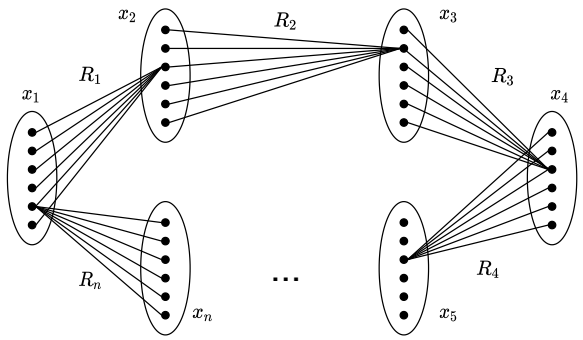

Figure 2: An n-cycle join.

Consider an n-cycle join

$$q = R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie \cdots \bowtie R_n(x_n, x_1),$$

shown in Figure 2, where each edge represents a tuple. More precisely, the instance $\mathbf{I}$ is constructed as follows. Each relation instance I_i has $\frac{N}{n}$ tuples. The tuples in each I_i have distinct values on x_i, while sharing the same value on x_{i+1} (define $x_{n+1} = x_1$). Note that $mf(x_i, I_i) = \frac{N}{n}$. No matter which join condition is removed, the elastic sensitivity is always

$$ES_q(\mathbf{I}) = \max_{0\leq k} e^{-\beta k} \tilde{LS}_q^{(k)}(\mathbf{I}) \geq \tilde{LS}_q^{(0)}(\mathbf{I}) = \left(\frac{N}{n}\right)^{n-1}.$$

For residual sensitivity, observe that for any $E \subseteq [n], T_E(\mathbf{I}) = 1$. So

$$\begin{aligned}
RS_q(\mathbf{I}) \leq &\max_{0\leq k\leq \frac{n-1}{1-e^{-\beta}}} e^{-\beta k} \hat{LS}_q^{(k)}(\mathbf{I}) \leq \hat{LS}_q^{(\frac{n-1}{1-e^{-\beta}})}(\mathbf{I}) \\
= &\max_{\mathbf{s}\in\mathbf{S}^{\frac{n-1}{1-e^{-\beta}}}} \max_{i\in[n]} \sum_{E'\subseteq[n]-\{i\}} \left(T_{n-E'-\{i\}}(\mathbf{I}) \prod_{j\in E'} s_j\right) \\
\leq &\max_{\mathbf{s}\in\mathbf{S}^{\frac{n-1}{1-e^{-\beta}}}} \max_{i\in[n]} \sum_{E'\subseteq[n]-\{i\}} \prod_{j\in E'} s_j \\
\leq &\max_{\mathbf{s}\in\mathbf{S}^{\frac{n-1}{1-e^{-\beta}}}} \max_{i\in[n]} \prod_{j\in[n]-\{i\}} (s_j + 1) \leq \left(\frac{n-1}{1-e^{-\beta}} + 1\right)^{n-1},
\end{aligned}$$

which is a constant.

5.3 Single Private Relation

When there is only one private relation, i.e., $n_P = 1$, all sensitivity measures collapse into one (see the remark at the end of Section 5.1). This means that the local sensitivity is already smooth, so we can simply use the local sensitivity to add noise following the

Cauchy/Laplace mechanism of [21]. This remains the best method if the released query answer needs to be unbiased.

Alternatively, the *truncation mechanism* [17, 27] has been shown to work well empirically, by trading some bias for lower variance (noise). The recent work of Tao et al. [27] presents the best solution in this category: They first compute the sensitivities of all tuples in the (only) private relation, then remove tuples with sensitivity above a certain threshold τ, and add noise calibrated to τ. They also show how to find a good τ in a differentially private manner. Thus, our method is the same as the first step in [27] in the case of a single private relation without foreign key constraints. After that, we may also use the truncation mechanism, which would result in the same result as in [27].

6 EXPERIMENTS

Our analytical results show that residual sensitivity is always no larger than elastic sensitivity, while there are queries and instances where the gap can be as large as $O\left(N^{n-1}\right)$. In this section, we conduct an experimental study on the actual gap on a collection of multi-way joins over both benchmark and real-world datasets. We also investigate its implications to the noise levels, as well as the computational overheads.

We have also tested wPINQ [23], the earliest work on multi-way join queries under the same DP policy as ours. Our results confirm those reported in [13], that wPINQ has worse utility than elastic sensitivity.

6.1 Setup

Datasets. We use two datasets in our experiments: TPC-H and the Facebook ego-network dataset. The TPC-H schema has many foreign key constraints. As our DP policy does not consider foreign key constraints, when a tuple is deleted in one relation, say `Customer`, we might obtain a neighboring instance that violates the foreign key constraint from `Orders` to `Customer`. To resolve this inconsistency, the correct way to interpret our DP policy is the following. We conceptually create the following relations by projecting the original relations to the join attributes. This results in the following 8 projected relations: `Region(RK)`, `Nation(RK, NK)`, `Customer(NK, CK)`, `Orders(CK, OK)`, `Supplier(NK, SK)`, `Part(PK)`, `PartSupp(SK, PK)`, `Lineitem(SK, PK, OK)`. We abbreviate these relations as `R`, `N`, `C`, `O`, `S`, `P`, `PS`, `L`, respectively. All relations except `R` and `P` capture relationships, e.g., `Orders(CK, OK)` stores which custom placed which order. We treat `C`, `O`, `S`, `PS`, `L` as private relations. There are no foreign key constraints between these private relations, so our DP policy will be well defined. Note that the same interpretation is used in the prior work [13]. We generated datasets with scale factors ranging from 0.01 to 10; the one with scale factor 1 contains about 7.5 million tuples.

The Facebook ego-network dataset is from SNAP [18], which contains 4,039 nodes and 176,467 directed edges. The nodes are organized as 193 "social circles". We merged these social circles into 5 "mega-circles", and created five relations $R_i(x, y), i = 1, \ldots, 5$, where each $R_i(x, y)$ contains all edges (x, y) that originate in the i-th mega-circle. In addition, we create a relation $R_6(x, y, z)$ that consists of all triangles formed by edges in R_1 or R_2, i.e., $R_6(x, y, z) := (R_1(x, y) \bowtie R_1(y, z) \bowtie R_1(z, x)) \cup (R_2(x, y) \bowtie R_2(y, z) \bowtie R_2(z, x))$. The 6 relations have 40968, 55125, 28231, 22486, 19179, 6080904 tuples, respectively, and they are all regarded as private relations. This data set models the scenario where different types of relations exist among the entities, such as friendship, co-workers, co-authors, family members, etc.

Queries. For TPC-H data, we used Q7, Q9, and Q5 from the benchmark, but removed projections, predicates, and group-by conditions, and their join structures are shown as q_1, q_2, q_3 in Figure 3. For the Facebook ego-network dataset, we used 5 queries, shown as $q_4, \ldots, q_8$ in Figure 3. Note that all joins on the TPC-H data are foreign-key joins (i.e., many-to-one), while those on the Facebook data are many-to-many joins.

6.2 Implementation

Both residual sensitivity and elastic sensitivity can be computed easily by SQL. Elastic sensitivity is computed by a UDF, after collecting the maximum frequencies of each relation by SQL [13]. Similarly, for residual sensitivity, each $T_E(\mathbf{I})$ can be computed using query (6). Then $RS_q(\mathbf{I})$ can be computed by formula (20) using a UDF. Thus, residual sensitivity enjoys the same benefit of elastic sensitivity that it can be easily integrated into any database system without any modification to the kernel. To automate this process, we have built a system prototype on top of PostgreSQL.[4] In our experiments, we used PostgreSQL 11.5. For wPINQ, we follow the similar settings as [13]. All experiments are conducted on a machine equipped with a 2.7 GHz Intel Core i7 and 16GB of memory.

However, we notice that PostgreSQL is not able to find the optimal query plan for executing query (6). Thus, we wrote another query rewriter that uses the following rules to rewrite a query in the form of (6). Our current query rewriter has only implemented a subset of these rules (this should be the job of the query optimizer of the database!), so one may still need some manual rewriting for the best performance.

Disconnected queries. If q_E consists of a few connected components, then q_E is the Cartesian product of the join results of each connected component. In this case, T_E is simply the product of the maximum boundaries of these connected components. It is more efficient to evaluate (6) for each connected component separately.

Example 6.1. Consider query q_1 in Figure 3 with $E = \{\mathsf{N}, \mathsf{C}, \mathsf{L}, \mathsf{S}\}$. We have $T_E = T_{E_1} \cdot T_{E_2}$ where $E_1 = \{\mathsf{N}, \mathsf{C}\}, E_2 = \{\mathsf{L}, \mathsf{S}\}$. □

Exploiting dependencies. Suppose there is a functional dependency $X \to Y$ and $X \subseteq \partial q_E$, then it is clear that we can add all attributes in Y to the group-by attributes ∂q_E without affecting the results of query (6). This step can be repeatedly applied. If all attributes of q_E can be added to ∂q_E, then the result of (6) must be either 1 (when q_E is non-empty) or 0 (when q_E is empty), provided there are no duplicated tuples in a relation.

Example 6.2. Consider query q_3 in Figure 3 with $E = \{\mathsf{R}, \mathsf{N}, \mathsf{S}, \mathsf{C}\}$, and $\partial q_E = \{\mathsf{SK}, \mathsf{CK}\}$. We have $\mathsf{SK} \to \mathsf{NK}$ (SK is the primary key of S) so we can add NK to ∂q_E. Also, $\mathsf{NK} \to \mathsf{RK}$ (NK is the primary key of N), so we can also add RK to ∂q_E. Now all attributes of q_E have been added to ∂q_E and we only need to check whether q_E is empty.

[4] Code is available at https://github.com/hkustDB/ResidualSensitivity.

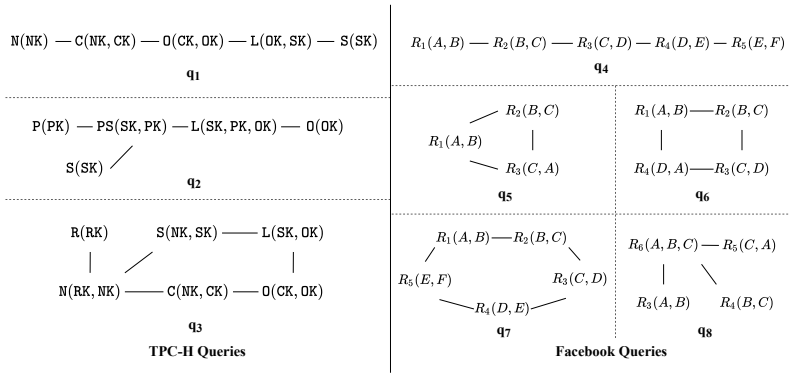

Figure 3: The join structure of queries.

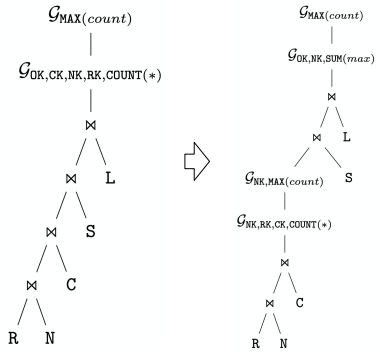

Figure 4: The rewriting of query of T_E for q_3 with $E = \{\mathsf{R}, \mathsf{N}, \mathsf{C}, \mathsf{S}, \mathsf{L}\}$.

Dataset		TPC-H			Facebook				
Query		q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8
Query result		6,001,215	6,001,215	239,917	1,666,978,389	19,927	285,754	6,348,654	21,613
wPINQ output		8,680	9,770	385	54.2	131.4	44.1	23.8	183.6
Residual Sensitivity (*RS*)	Min Value	694	694	49	77,100,000	203	2,790	86,800	50
	Max Value	51,900	52,000	51,800	301,000,000	1,410	54,500	2,050,000	51,300
	Running Time(s)	27.6	53.6	48.6	2.96	1.68	4.71	18.1	20.1
Elastic Sensitivity (*ES*)	Min Value	1,740	2,140	175,000,000	25,500,000,000	219,000	110,000,000	55,000,000,000	561
	Max Value	4,950,000	1,870,000	263,000,000	25,500,000,000	219,000	110,000,000	55,000,000,000	2,440,000
	Running Time(s)	7.55	9.02	6.73	0.300	0.611	0.628	5.725	8.78
ES/*RS*	Min	2.51×	3.08×	5,069×	84.8×	156×	2,010×	26,900×	11.2×
	Max	273×	67×	3,580,000×	330×	1,080×	39,300×	634,000×	178×
	Avg	94.4×	27.8×	1,750,000×	286×	875×	27,300×	503,000×	80.6×
Running time: *RS*/*ES*		3.65×	5.94×	7.23×	9.86×	2.75×	7.50×	3.17×	2.29×

Table 2: Comparison among wPINQ, residual sensitivity and elastic sensitivity.

Aggregation push-down. When evaluating (6), PostgreSQL would first compute the join and then the aggregation, which is inefficient. We can push down the aggregation as far as possible to reduce the execution cost. Joglekar et al. [12] present a general framework for pushing down aggregations, which in turn defines the width w as mentioned in Section 4.1.

Example 6.3. Consider query q_3 in Figure 3 with $E = \{\mathsf{R}, \mathsf{N}, \mathsf{C}, \mathsf{S}, \mathsf{L}\}$, we can first add attributes NK, RK to ∂q_E by exploiting dependencies. Then, we rewrite (6) by pushing down both COUNT and MAX. The query plans before and after this rewrite are shown in Figure 4. The optimized query plan has $w = 1$ and it can be evaluated in linear time. In practice, this reduces PostgreSQL's execution time by roughly 100 times in our experiments. □

Incremental computation. Finally, observe that the subquery in (6):

$$\texttt{SELECT COUNT}(*)\ \texttt{AS Boundary FROM}\ q_E\ \texttt{GROUP BY}\ \partial q_E \tag{32}$$

can be computed incrementally for E's that differ by one relation. Thus, we can order the computation of T_E's in a way so that previous results can be re-used.

Example 6.4. Consider query q_6 in Figure 3, where we need to compute T_{E_1}, T_{E_2}, T_{E_3} for $E_1 = \{R_1\}$, $E_2 = \{R_1, R_2\}$, $E_3 = \{R_1, R_2, R_3\}$, among others. First, the result of executing (32) on q_{E_1} is just R_1 itself, with an additional column Boundary whose values are all 1. Then we execute (32) on q_{E_2}:

$$\texttt{SELECT A, C COUNT}(*)\ \texttt{AS Boundary FROM}\ R_1, R_2 \\ \texttt{WHERE}\ R_1.\texttt{B} = R_2.\texttt{B GROUP BY A, C} \tag{33}$$

Next, we can run

$$\texttt{SELECT A, D SUM((33).Boundary) AS Boundary FROM (33)},\ R_2 \\ \texttt{WHERE (33).C} = R_2.\texttt{C GROUP BY A, D}$$

to obtain the result of (32) on q_{E_3}. □

6.3 Experimental Results

Compare with wPINQ. We test wPINQ on all queries and find it losses the utility in all cases: as shown in Table 2, wPINQ outputs a result much less than 1% of the real query result. That is because, as shown in Section 2, the wPINQ ensures any tuple can at most effect one counting query result by scaling down the tuples' weights. Such operations can lead to the output result much smaller than the real one and that becomes more problem with the number of relations increase. The wPINQ performs well in the case where most tuples affect no more than one query result or the case where

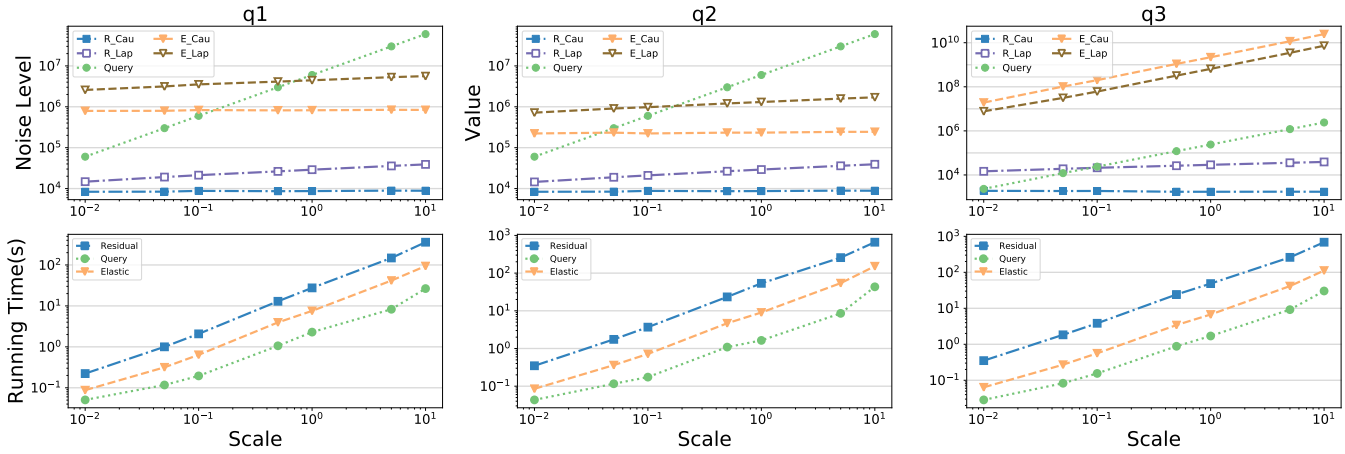

Figure 5: Running times and noise levels of residual sensitivity and elastic sensitivity with different noise mechanisms for different queries and data scales. *R/E* represents the noise level calculated from residual sensitivity or elastic sensitivity, respectively, while *Cau/Lap* denotes the respective noise mechanism.

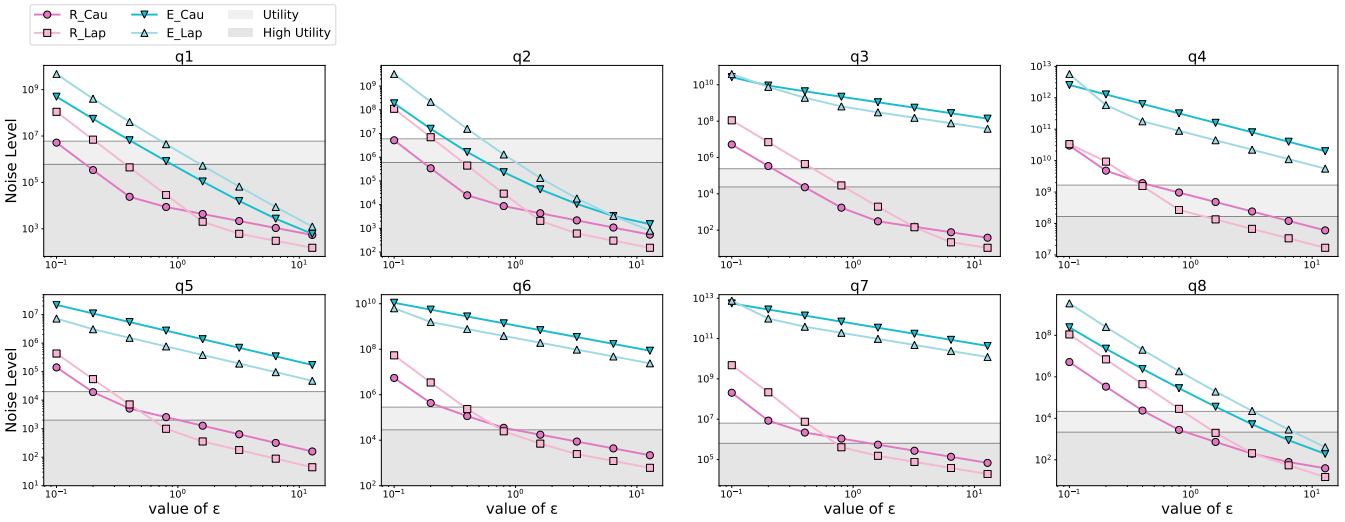

Figure 6: The noise level of residual sensitivity and elastic sensitivity for various values of ε.

tuples in the same relation have the same degree(counting triangles incident on vertices with fixed degrees). However, in our experiments, tuples join with any number of tuples. Besides, it has been shown that wPINQ performs much worse than elastic sensitivity for non-histogram counting queries [13].

Compare with elastic sensitivity. Since elastic sensitivity (*ES*) and residual sensitivity (*RS*) can be used in exactly the same manner to calibrate noise, where the noise level is proportional to the sensitivity value, we can compare their sensitivity values directly, given the same smoothing parameter β. We tried 7 different values $\beta = 0.01,0.02,0.04,0.08,0.16,0.32,0.64$, and computed *ES* and *RS* for each of the queries in Figure 3. For TPC-H queries, we used the dataset with scale factor 1. In Table 2, we report the maximum, minimum, and the average value of the sensitivities over the β's, as well as their ratios. Note that both sensitivities decrease as β increases, but the ratio ES/RS is not necessarily monotone.

First, the experimental results confirm our theoretical analysis: *RS* is always no more than *ES*, while the gap can be *very* large for certain queries. More importantly, the results reveal two fundamental reasons why *RS* offers a much tighter upper bound on the smooth sensitivity than *ES*. The first one is the data skewness. *ES* is calculated based on multiplying the maximum frequencies in the relations. In doing so, it makes the simple but very pessimistic assumption that these most frequent attribute values can join, thereby shooting up the sensitivity tremendously. If data is skewed, there are very few heavy hitters, and the chance that they can join is low. On the other hand, *RS* computes the actual join of the residual query, so the presence of the heavy hitters will not affect the sensitivity,

unless they can actually join. We see from Table 2 that the ratio ES/RS is generally larger on the Facebook dataset, which is real network data with high skewness. On the other hand, TPC-H data is more uniform. The second situation where RS is much smaller than ES is on cyclic queries (q_3, q_5, q_6, q_7). This is because the approach towards cyclicity taken by ES, which just removes some join conditions, is too simplistic. Removing these join conditions dramatically enlarges the sensitivity, because these join conditions put restrictions on how join results can be formed. On the other hand, RS is defined in a unified manner over acyclic and cyclic queries.

In Table 2, we also report the running times of these queries, which include executing the query itself plus the time spent in computing the sensitivity. The time to add noise to the query answer is negligible. We repeated each query 10 times and the average running time is reported. We see that RS indeed requires more time to evaluate, but considering the huge improvement in the accuracy of the released query answers (which will be more prominently compared next), the extra time is well spent.

Scalability. To examine the effects as data scale changes, we used TPC-H datasets with scale factors ranging from 0.01 to 10. To get a more intuitive sense, we calculated the noise level from the computed sensitivity with both the Cauchy mechanism and Laplace mechanism as described in Section 3.5. We fix the privacy parameter $\varepsilon = 0.8$. For the Laplace mechanism, we set $\delta = 10^{-7}, 2 \times 10^{-8}, 10^{-8}, 2 \times 10^{-9}, 10^{-9}, 2 \times 10^{-10}, 10^{-10}$ for TPC-H dataset with different scale and set $\delta = 10^{-9}/10^{-7}$ for Facebook dataset with R_6 involved/uninvolved.

The noise levels for different data scales are plotted in the top row of Figure 5, in which we also plot the actual query answer. Note that a noise level higher than the query answer means that the noise-masked result would be basically useless. We see from the results that the noise level from RS is always below the query answer, while this is not the case for ES. In particular, for q_3, which is a cyclic query, the noise level of ES is always (much) higher than the query answer.

Another observation is that the noise level does not increase much with the data scale. Intuitively, this is because sensitivity measures the impact of an individual tuple, which in some sense is a "local" measure. The implication is that the released query answer is more accurate, relatively speaking, on larger datasets, which is a nice property. We also observe that the noise level from the Laplace mechanism increases more than that from Cauchy. This is because we set smaller δ for larger dataset in the Laplace mechanism, which can further bring impact on β, while β is independent of data size for Cauchy.

The running times are plotted in the second row of Figure 5. There is not much surprise there, but it is nice to see that the growth rate of the time for computing the RS is more or less the same as that computing the query itself. This means that the cost of evaluating query (6) (after appropriate rewriting) is in the same ballpark as that of executing the query itself, while RS needs to evaluate a constant number of such queries.

Privacy parameter ε. Lastly, we conducted experiments to see how the privacy parameter ε affects various mechanisms. Recall that a smaller ε means higher privacy protection, but also increases the noise level. In fact, it does so via two channels: (1) A smaller ε increases the coefficient of the noise distribution; and (2) a smaller ε leads to a smaller β, hence a higher sensitivity. We tried various values of ε from 0.1 to 12 and tested the 8 queries. The results are plotted in Figure 6. In the figures, we also plot the query answer and 10% of that: A noise level below the query answer is considered to have utility while having high utility if it is below 10% of the query answer.

The first message the figures convey is nothing but a reconfirmation of the results in Table 2: The gap between the sensitivities directly translates to that between the noise levels. On cyclic queries and/or data with high skewness (q_3, q_4, q_5, q_6, q_7), ES does not have utility even with ε is as large as 12, which is usually considered too high. For easy queries (q_1, q_2, q_8), RS is able to obtain utility or high utility at a much smaller ε.

The more interesting observation is the comparison between the two noise mechanisms. We see that, when combined with RS, the two mechanisms have a crossover on every query: the Cauchy mechanism seems to work better for smaller ε, while the Laplace mechanism favors larger ε. This is precisely due to the two channels through which ε affects the noise level. The coefficients of both noise distributions are inversely proportional to ε, so the noise levels both decrease as ε increases. Meanwhile, ε also affects β through the second channel, but its impact on β is larger for the Laplace mechanism. A larger β reduces RS, although this relationship is a complicated one. Anyhow, because the Laplace mechanism is more sensitive to the change in ε, we see steeper curves in its noise levels as we vary ε. On the other hand, this phenomenon is not obvious for ES. This is because ES is not very sensitive to β on many queries (as can be seen from Table 2), which reduces the effects of the second channel. In fact, being sensitive to β is a necessary property of any good upper bound on the smooth sensitivity, which itself is highly sensitive to β. In the extreme case, if an upper bound is completely insensitive to β, it just boils down to the global sensitivity.

7 FUTURE WORK

We mention three interesting directions for further investigation. The first is how to support self-joins. The second is how to release many join queries in a way better than direct composition. There is a lot of work in this direction for queries without joins, but the problem is wide open for joins. Finally, the *complex DP policy with foreign key constraints* [17, 27] can provide privacy protection at the entity level, but the existing methods only support one private entity relation. How to extend to multiple private entity relations remains a challenging problem.

ACKNOWLEDGMENTS

This work has been supported by HKRGC under grants 16202317, 16201318, 16201819, and 16205420. We would also like to thank the anonymous reviewers who have made valuable suggestions on improving the presentation of the paper.

REFERENCES

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

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

[3] Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Satyen Kale, Frank McSherry, and Kunal Talwar. 2007. 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*. 273–282.

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

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

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

[7] Xiaofeng Ding, Xiaodong Zhang, Zhifeng Bao, and Hai Jin. 2018. Privacy-preserving triangle counting in large graphs. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 1283–1292.

[8] Cynthia Dwork and Jing Lei. 2009. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 371–380.

[9] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284.

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

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

[12] Manas R Joglekar, Rohan Puttagunta, and Christopher Ré. 2016. Ajar: Aggregations and joins over annotated relations. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 91–106.

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

[14] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. 2011. Private analysis of graph structure. *Proceedings of the VLDB Endowment* 4, 11 (2011), 1146–1157.

[15] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2013. Analyzing graphs with node differential privacy. In *Theory of Cryptography Conference*. Springer, 457–476.

[16] Daniel Kifer and Ashwin Machanavajjhala. 2011. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 193–204.

[17] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: a differentially private SQL query engine. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1371–1384.

[18] Jure Leskovec and Andrej Krevl. 2016. SNAP datasets: Stanford large network dataset collection (2014). *URL http://snap. stanford. edu/data* (2016), 49.

[19] Frank D McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 19–30.

[20] Arjun Narayan and Andreas Haeberlen. 2012. DJoin: Differentially private join queries over distributed databases. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*. 149–162.

[21] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2007. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. 75–84.

[22] Catuscia Palamidessi and Marco Stronati. 2012. Differential Privacy for Relational Algebra: Improving the Sensitivity Bounds via Constraint Systems. In *QAPL*.

[23] Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2014. Calibrating Data to Sensitivity in Private Data Analysis. *Proceedings of the VLDB Endowment* 7, 8 (2014).

[24] Wahbeh Qardaji, Weining Yang, and Ninghui Li. [n.d.]. Practical differentially private release of marginal contingency tables. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of Data*. 1435–1446.

[25] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1954–1965.

[26] Vibhor Rastogi, Michael Hay, Gerome Miklau, and Dan Suciu. 2009. Relationship privacy: output perturbation for queries with joins. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 107–116.

[27] Yuchao Tao, Xi He, Ashwin Machanavajjhala, and Sudeepa Roy. 2020. Computing Local Sensitivities of Counting Queries with Joins. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 479–494.

[28] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. 2010. Differential privacy via wavelet transforms. *IEEE Transactions on knowledge and data engineering* 23, 8 (2010), 1200–1214.

[29] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2015. Private release of graph statistics using ladder functions. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 731–745.

[30] Xiaojian Zhang, Rui Chen, Jianliang Xu, Xiaofeng Meng, and Yingtao Xie. 2014. Towards accurate histogram publication under differential privacy. In *Proceedings of the 2014 SIAM international conference on data mining*. SIAM, 587–595.

A A QUASI-POLYNOMIAL TIME ALGORITHM FOR COMPUTING SMOOTH SENSITIVITY

In this section, we describe an $N^{O(\log N)}$-time algorithm for computing $SS_q(\mathbf{I})$.

Recall the definition of $SS_q(\mathbf{I})$:

$$SS_q(\mathbf{I}) = \max_{k \geq 0} e^{-\beta k} LS_q^{(k)}(\mathbf{I}).$$

First, we show that it is sufficient to consider $k = 0, 1, \ldots, \frac{2(n-1)}{\beta} \ln N$. It is trivial to see, $LS^{(k)}(\mathbf{I}) \leq (N+k)^{n-1}$. When $k \geq \frac{2(n-1)}{\beta} \ln N$, we have $e^{-\beta k} LS_q^{(k)}(\mathbf{I}) \leq e^{-\beta k}(N+k)^{n-1} \leq 1$, so it has no effects on the max.

Then, for each k, we need to compute

$$LS_q^{(k)}(\mathbf{I}) = \max_{\mathbf{I}' \in \mathcal{I}^k} LS_q(\mathbf{I}'),$$

where $\mathcal{I}^k = \{I' : d(\mathbf{I}, \mathbf{I}') = k\}$. The domains of the attributes can be infinite, thus $\mathcal{I}^k$ is also infinite. To resolve this issue, we show that we do not need to consider the entire domains of the attributes, while some finite sub-domains are sufficient for computing $LS_q^{(k)}$.

For a given instance $\mathbf{I}$ and any attribute x, the *active domain* of x is $\mathbf{dom}_{act}(x) = \cup_{i \in [n]} \pi_x I_i$. For $\mathbf{x} = (x_1, \ldots, x_k)$, let $\mathbf{dom}_{act}(\mathbf{x}) = \mathbf{dom}_{act}(x_1) \times \cdots \times \mathbf{dom}_{act}(x_k)$. Tao et al. [27] show that when computing $LS_q(\mathbf{I})$, only neighboring instances $\mathbf{I}, \mathbf{I}'$ that differ by a tuple $t \in \mathbf{dom}_{act}(\mathbf{x}_i), i \in P$ need to be considered. However, when computing $LS_q^{(k)}$, we cannot only consider $\mathbf{I}'$ that differ from $\mathbf{I}$ by k tuples in $\mathbf{dom}(\mathbf{x}_i), i \in P$. The reason is that, these k tuples may correlate with each other through values not in the active domains. To solve this issue, we add k values to $\mathbf{dom}_{act}(x)$ to form the *k-extended active domain* for each attribute x:

$$\hat{\mathbf{dom}}_{act}^{k}(x) = \mathbf{dom}_{act}(x) \cup \{a_1, \ldots, a_k\},$$

where $a_1, \ldots, a_k$ are k different values in $\mathbf{dom}(x) - \mathbf{dom}_{act}(x)$. Similarly, let $\mathbf{dom}_{act}(\mathbf{x}) = \mathbf{dom}_{act}(x_1) \times \cdots \times \mathbf{dom}_{act}(x_k)$ for $\mathbf{x} = (x_1, \ldots, x_k)$. We claim that when computing $LS_q^{(k)}$, it is sufficient to consider

$$\mathcal{I}^k = \{\mathbf{I}' : \mathbf{I}, \mathbf{I}' \text{ differ by } k \text{ tuples in } \hat{\mathbf{dom}}_{act}^{k}(\mathbf{x}_i)\}.$$

Indeed, suppose $\mathbf{I}'$ has tuples have value a' on attribute x and $a' \notin \hat{\mathbf{dom}}_{act}^{k}(x)$. Because $\mathbf{I}'$ has at most k tuples not in $\mathbf{I}$, there must be an extra a_i in $\hat{\mathbf{dom}}_{act}^{k}(x)$ that is not used. Then we can remap a' to a_i, which does not induce any structural change in $\mathbf{I}'$. Note that the size of $|\mathcal{I}^k|$ is at most $O((N+k)^{mk})$, where m is the number of attributes. This is $N^{O(\ln N)}$ in terms of data complexity.

Then, we can calculate the $SS_q(\mathbf{I})$ as follows. For each $k = 0, 1, \ldots, \frac{2(n-1)}{\beta} \ln N$, we compute $LS_q^{(k)}(\mathbf{I})$. To compute $LS_q^{(k)}(\mathbf{I})$, we enumerate all $\mathbf{I}' \in \mathcal{I}^k$, and use Theorem 4.5 to calculate $LS_q(\mathbf{I}') = \max_{i \in P} T_{[n]-\{i\}}(\mathbf{I}')$, hence $LS_q^{(k)}(\mathbf{I})$. Finally, we obtain $SS_q(\mathbf{I})$ by taking the maximum of $e^{\beta k} LS_q^{(k)}(\mathbf{I})$ over all k. The running time is still $N^{O(\ln N)}$.