# Differentially private instance-based noise mechanisms in practice

Sébastien Canard, Baptiste Olivier and Tony Quertier
{sebastien.canard, baptiste.olivier, tony.quertier}@orange.com

June 1, 2017

**Abstract**

Differential privacy is a widely used privacy model today, whose privacy guarantees are obtained to the price of a random perturbation of the result. In some situations, basic differentially private mechanisms may add too much noise to reach a reasonable level of privacy. To answer this shortcoming, several works have provided more technically involved mechanisms, using a new paradigm of differentially private mechanisms called *instance-based noise mechanisms*.

In this paper, we exhibit for the first time theoretical conditions for an instance-based noise mechanism to be $(\epsilon, \delta)$-differentially private. We exploit the simplicity of these conditions to design a novel instance-based noise differentially private mechanism. Conducting experimental evaluations, we show that our mechanism compares favorably to existing instance-based noise mechanisms, either regarding time complexity or accuracy of the sanitized result. By contrast with some prior works, our algorithms do not involve the computation of all local sensitivities, a computational task which was proved to be NP hard in some cases, namely for statistic queries on graphs.

Our framework is as general as possible and can be used to answer *any* query, which is in contrast with recent designs of instance-based noise mechanisms where only graph statistics queries are considered.

## 1 Introduction

### 1.1 Context and related work

One big concern in data publishing is the privacy of the individuals involved in these data. As the opportunities and the means to release useful information from individual data (*a.k.a.* personal data) grow wider, the leakage of information threatens more and more these individuals. That is the reason why researchers have proposed several rigorous notions of privacy in the last few years and, among them, one of the most promising is *differential privacy*. This notion, more precisely referred as $\epsilon$-differential privacy, was introduced by

Dwork, McSherry, Nissim and Smith in [7]. It provides strong guarantees that a mechanism $\mathcal{A}$ is privacy-preserving at a security level depending on $\epsilon$, but not depending on arbitrary side information owned by an adversary. Informally speaking, a differentially private mechanism ensures that any of its outputs is essentially likely to occur, independently of the presence or absence of a single individual in the database.

Consider a dataset $x$ and a query $f$ on this dataset. A common way to design an $\epsilon$-differentially private randomized mechanism $\mathcal{A}$ is to add noise to the query output $f(x)$ as: $\mathcal{A}(x) = f(x) + Z$, where $Z$ is some well-chosen random variable (independent of $x$). It may happen that some situations require noise which cannot provide $\epsilon$-differential privacy, but satisfy some weaker notions of privacy. The most known and widely used weakened notion was introduced in [6] and is called $(\epsilon, \delta)$-differential privacy, where $\delta$ is an additional privacy parameter. An important family of $(\epsilon, \delta)$-differentially private mechanisms is given by *instance-based noise* mechanisms [16] which take the following form: $\mathcal{A}(x) = f(x)+Z_x$ for some random variable $Z_x$ depending on the queried dataset $x$. In [16], the differentially private mechanism is calibrated to a new kind of sensitivity called *Smooth Sensitivity*, while previous algorithms were always designed with respect to global sensitivity (see Section 2 for formal definitions). This careful look at sensitivity results in better accuracy of differentially private algorithms, while almost keeping the same level of privacy. Such schemes are now widely used, in particular to release differentially private (statistics of) graphs (see [2], [4], [10], [11], [15], [17], [18]).

The main drawback of Smooth Sensitivty algorithms is time complexity: in particular, the authors of [11] show that the computation Smooth Sensitivity for countings of $k$-triangles in graphs is a NP-hard task (when edge-privacy is considered). As an alternative, they designed an instance-based noise mechanism for such queries, that was differentially private and did not rely on the computation of Smooth Sensitivity. Since then, a few new frameworks were introduced to replace Smooth Sensitivity technique in the context of private subgraph countings: recursive mechanism from [4], analysis on structured graphs from [2], ladder functions framework in [18].

Our main contribution in the current paper is a new design of instance-based noise differentially private algorithms. As a first step, we exhibit simple conditions for an instance-based noise mechanism to be $(\epsilon, \delta)$-differentially private. Then we take advantage of these conditions to design a novel instance-based noise mechanism. Our new design is in a similar spirit as Smooth Sensitivity mechanism, but with a sharp difference: our mechanism requires only the computation of the largest local sensitivities, and not the computation of all local sensitivities.

We also provide an algorithm for practical implementations, and we compare it to state-of-the-art concurrent techniques. Our algorithm provides $(\epsilon, \delta)$-differential privacy, and is easy to handle in practice, in all cases where privacy-preserving queries are asked on a database. The algorithm drastically outperforms naive methods to answer a query in a differentially private manner, and also most of the refined techniques such as Smooth Sensitivity mechanism, in terms of an advantageous trade-off between utility and time complexity.

## 1.2 Details on our contributions

Here we summarize our contributions:
• We give in Theorem 8 simple conditions for an instance-based noise mechanism to be $(\epsilon, \delta)$-differentially private.
• We use conditions from Theorem 8 to design a new instance-based noise mechanism, that we call Largest Local Sensitivities mechanism (LLS in the sequel) which is $(\epsilon, \delta)$-differentially private and allows for drastically reducing noise compared to standard Laplacian noise mechanism. LLS only needs to compute (or approximate) a restricted number of local sensitivities, and not all, which is in contrast with the so-called Smooth Sensitivity.
• We give an analysis of LLS in terms of time complexity and accuracy on some examples, either theoretical or on real-life public datasets. In particular, we will show that LLS is particularly attractive when datasets with low local sensitivities are queried. We also compare LLS to the most important examples of instance-based noise mechanisms already appeared in the literature: Smooth Sensitivity technique from [16], Noisy Local Sensitivities from [11], and Ladder Functions Mechanism from [18].

## 1.3 Organization of the paper

In section 2, we recall relevant notions of differential privacy. Section 3 is devoted to the design of LLS. In section 4, we analyse time complexity and accuracy of LLS . In section 5 we make a theorical comparison between LLS and other instance-based noise mechanisms. In section 6, we give experimental results of LLS and other instance-based noise mechanisms on different real-world dataset. Missing proofs of theoretical results and further details are postponed to the Appendix 8.

# 2 Preliminaries

In this section, we recall the relevant notions of differential privacy we will need in the sequel. We refer to [5] and [8] for basic notions about differential privacy. We use the notation $\mathbb{P}(\mathsf{event})$ to denote the probability that $\mathsf{event}$ occurs.

## 2.1 Privacy model

We consider a simple scenario for data sanitization where a *sanitizer* aims at publishing results of queries over the data he owns, while preserving the anonymity of some sensitive information in the data. The privacy guarantee he wants to achieve is differential privacy, which asserts the secrecy (with high probability) of the participation of any single user in the sanitizer's database.
More formally, we will denote by $\mathcal{D}$ the dataspace, that is the set of all possible

datasets owned by the sanitizer. Given a query $f$ defined on $\mathcal{D}$ and datasets $x_i \subset \mathcal{D}$, the sanitizer's goal is to release a differentially private versions of the values $f(x_i)$. To do so, a sanitization randomized mechanism $\mathcal{A} : \mathcal{D} \to \mathbb{R}$ is applied to release close perturbations of $f(x_i)$, while satisfying differential privacy guarantee.

## 2.2 Sensitivity of a query

Differential privacy relies on a relation of neighboring that is defined as follows.

**Definition 1** Let $\mathcal{D}$ be the dataspace. Two datasets $x, x' \subset \mathcal{D}$ are said to be *neighbor* if they differ from each other by a single individual's data. In that case we denote $x \sim x'$.
We say that the dataspace $\mathcal{D}$ is *connected* if for all $x, x' \subset \mathcal{D}$, there exists a sequence $(x_i)_{1 \leq i \leq N}$ such that $x_1 = x$, $x_N = x'$ and

$$x_i \sim x_{i+1} \text{ for all } 1 \leq i \leq N-1.$$

To simplify the exposition of our techniques, we will always assume that the dataspace $\mathcal{D}$ is connected. In fact, this assumption can be removed in our context by performing parallel sanitization mechanisms on the connected components of $\mathcal{D}$ (see for instance Theorem 4 in [13]).

**Example 2** For data tables, $\mathcal{D}$ is a concatenation of rows, where each row is the data of a single individual. Then, we have $x, x' \subset \mathcal{D}, x \sim x'$ if $x$ and $x'$ differ from a single row.

Any differentially private mechanism calibrates the amplitude of its random perturbation to a quantity called *sensitivity*. The latter depends on the output to be sanitized, namely the query $f$ and the queried dataspace $\mathcal{D}$, and on the privacy guarantee, which is defined by the neighboring relation for the case of differential privacy.
In the current paper, we exclusively deal with real-valued queries $f : \mathcal{D} \to \mathbb{R}$. For a multi-dimensional range $\mathbb{R}^d$, that is a *workload* of real-valued queries $(f_i)_{1 \leq i \leq d}$, our results can be straightforwardly extended using composition theorems (see Chapter 3.5 in [8]). Sensitivity is defined as follows in the one-dimensional case.

**Definition 3 (Global sensitivity)** Let $f : \mathcal{D} \to \mathbb{R}$ be a query. The global sensitivity $GS(f)$ of $f$ (denoted $GS$ if there is no confusion) is defined by

$$GS(f) = \sup_{x \sim x', x, x' \subset \mathcal{D}} |f(x) - f(x')|.$$

While global sensitivity depends only on $\mathcal{D}$ and not on a particular dataset $x$, it is considered as a public value. By contrast, the *local sensitivity* around $x$

defined below depends on a particular instance $x$, and so is a private information regarding the issue of masking the participation of a single user.

**Definition 4 (Local sensitivity)** For all $x \subset \mathcal{D}$, the local sensitivity of a query $f : \mathcal{D} \to R$ at $x$, denoted by $LS(f)(x)$ (or simply $LS(x)$), is defined by

$$LS(f)(x) = \sup_{x \sim x'} |f(x) - f(x')|.$$

**Example 5** Let us introduce now a basic example that will be used to illustrate our techniques along the paper. The following dataset contains the average annual salary for different classes of employees from a large company:

$$\begin{aligned}
&x_5 = \{\ 24634, 29475, 37468, 48104, 69624, 113511\ \}\\
&x_k \text{ the dataset of the first } k \text{ salaries in } x_5\\
&\text{and } \mathcal{D} = \{\ x_k \mid 1 \leq k \leq 5\ \}.
\end{aligned}$$

We define a neighboring relation as follows: $x \sim x' \Leftrightarrow x = x_i,\ x' = x_j$ and $i \neq j \in \{k, k+1\}$ for some $1 \leq k < 5$. We consider a simple query that sums all the salaries in the queried dataset, that is $f(x) = \sum_{x^{(i)} \in x} x^{(i)}$ for all $x \subset \mathcal{D}$. Note that there are real-life situations in which the salary value is not a sensitive information on its own, whereas the association of a specific salary to a single employee is. Once the dataspace $\mathcal{D}$ is fixed, the value $GS(f)$ is public and does not depend on a queried dataset $x \subset \mathcal{D}$: the pair $(\mathcal{A}(x), \Delta)$ leaks information about the participation of a single individual only through the value $\mathcal{A}(x)$. By contrast, the pair $(\mathcal{A}(x_1), LS(f)(x_1))$ would reveal that $x_1$ was the queried dataset, no matter the value of $\mathcal{A}(x_1)$: $LS(f)(x_1) = 29475$ completely reveals the class of employees at stake, that is those with salary 24634.

## 2.3 Differential privacy and Laplacian mechanisms

The following notion of $(\epsilon, \delta)$-differential privacy is a generalization of the so-called $\epsilon$-differential privacy. More precisely, the latter notion is obtained from the former by taking $\delta = 0$.

**Definition 6** (**$(\epsilon, \delta)$-differential privacy** [6]). Let $\mathcal{D}$ be a database. A randomized algorithm $\mathcal{A} : \mathcal{D} \to \mathbb{R}$ is $(\epsilon, \delta)$-*differentially private* if, for all subsets $S \subset \mathbb{R}$, and for all $x \sim x'$, we have

$$\mathbb{P}(\mathcal{A}(x) \in S) \leq e^{\epsilon}\ \mathbb{P}(\mathcal{A}(x') \in S) + \delta.$$

In the current paper, differential privacy is achieved using Laplacian mechanisms. These randomized mechanisms are defined as follows:

$$\mathcal{A}(x) = f(x) + Z_x,$$

where $Z_x$ is a Laplacian random variable of parameter $\lambda_x$, possibly depending on the queried instance $x \subset \mathcal{D}$. The most straightforward way to achieve $\epsilon$-differential privacy uses a uniform parameter $\lambda = \lambda_x$ equal to $\frac{\epsilon}{GS(f)}$. Then the corresponding Laplace distribution is $g(y) = \frac{1}{\sqrt{2}\lambda} e^{-\lambda y}$.
As early noticed in [16], much more accurate algorithms can be obtained when exploiting the dependence on $x$ in the random variable $Z_x$, to the price of a small loss in privacy ($(\epsilon, \delta)$-differential privacy instead of $\epsilon$-differential privacy). Our goal si to introduce new *instance-dependent* based algorithms, and to show their practicability through experiments on real datasets.

## 2.4 Privacy of local sensitivities, and their computation in practice

When using (standard) Laplacian mechanism, the global sensitivity value $GS(f)$ should be reasonably considered as a public value. Indeed, if many queries are to be made, global sensitivity can be easily approximated: it is deduced from the amount of noise $1/\lambda$ and the public parameter $\epsilon$, where the former can itself be estimated if a single dataset is queried several times. In the contrary and as shown by Example 5, the value of the local sensitivity used to perform an instance-based noise mechanism cannot be public. When a query result $f(x)$ is asked to the sanitizer, the latter is allowed to publish $\mathcal{A}(x)$ and $GS(f)$, but in no case the corresponding sensitivity value $LS(f)(x)$. We believe that $GS(f)$ should also be a public value in the instance-based noise case, and we assume so in the current paper.
In practice, the sanitizer faces the problem of estimating a correct value for sensitivities $LS(f)(x), x \subset \mathcal{D}$ (and even that of $GS(f)$ sometimes). As will be explained below and along the paper, the exact computation (or even an accurate approximation) of some sensitivities is sometimes infeasible. Moreover, no lower bound on sensitivities should be used, since it would break the differential privacy guarantee. Sanitizer's goal is then to provide the most accurate upper bound on the sensitivity considered.
*Theoretical* sensitivity comes with a formula (given in Definition 4), which is computable in some cases but hard to accurately estimate in other cases. Indeed, the computation depends on the triple $(\mathcal{D}, f, \sim)$ which may induce hard computations. As shown in [16], this is the case when $\mathcal{D}$ is a set of graphs for specific queries. Hard computations can also occur when $\mathcal{D}$ is simply a collection of data tables: local sensitivities values are allowed to be either public as in Example 14 (while the correspondence with individuals is kept private) or private. In both cases, an exhaustive computation of pairs $(x, LS(f)(x))$ is not achievable in reasonable time for many large databases, as illustrated in Example 7.
Many efforts have been made in previous works ([4], [2], [18]) to propose instance-based noise differentially private mechanisms, with alternative methods to avoid computing all local sensitivities. Our paper follows this line of research with a new alternative, allowing to compute only the approximations of the largest

sensitivities.

**Example 7** Let us build a new query scenario based on Example 5. Assume now that each salary is a (known) function $g$ of several other attributes: age, salary of previous job... Then each dataset $x$ can be described as a data table whith the following configuration: line $i$ in the table corresponds to a single individual, and a line format is a tuple of the form $x_i = (g(x_i^1, x_i^2 ...), x_i^1, x_i^2 ...)$ which describes the attributes (salary, age, previous salary,...) of the individual. Assume also that our sanitizer owns a large database, and that we allow many distinct salary values (by refining the precision of salaries to define the different *classes* of employees). This results in a large range of possible values $(LS(f)(x))_{x \in \mathcal{D}}$, together with a large domain $\mathcal{D}$. Indeed, the size of $\mathcal{D}$ directly depends on the range of all attribute values in the data table, that is on possible variations of values $(x_i^j)_{i,j}$.
For sufficiently many attribute values and/or sufficiently large ranges for these attributes, the sanitizer will not be able to estimate all pairs $(x, LS(f)(x))$, nor to store them with efficient time and space constraints. Thus a sanitization mechanism requiring only the computation of a *restricted* subset of sensitivities would greatly simplify the sanitizer's task.

## 3 Design of a $(\epsilon, \delta)$-differentially private instance-based noise mechanism

Different mechanisms providing differential privacy exist in the literature and, among them, the instance-based noise [16] is one of the most promising one. In this section, we describe our novel algorithm, namely LLS.

### 3.1 Instance-based noise and $(\epsilon, \delta)$-differential privacy

The first purpose of this section is to show that the instance-based noise technique introduced in [16] can be generalized, and satisfies privacy in the sense of $(\epsilon, \delta)$-differential privacy.
The idea from [16] is to consider a mechanism of the form

$$\mathcal{A}(x) \quad = \quad f(x) + \frac{1}{\lambda_x} Z, \tag{1}$$

where $Z$ is a Laplacian random variable with parameter 1, and the instance-based coefficient $\frac{1}{\lambda_x}$ defines the noise magnitude. With such a mechanism, one can hope for a reduced error, that is larger values of $\lambda_x$ for subsets $x \subset \mathcal{D}$ which are less sensitive. However, some constraints on the values $\lambda_x$ are required for $\mathcal{A}$ to satisfy privacy guarantees. We will also make use of the following notation

$$\Delta_{x,x'} = \Delta_{x,x'}(f) = |f(x) - f(x')|.$$

The following Theorem 8 gives simple conditions for a mechanism as above to be $(\epsilon, \delta)$-differentially private. These conditions will be used in the next section to design algorithm LLS.

**Theorem 8** *Let $\mathcal{A}$ be the mechanism defined by Equation (1), such that the noise magnitude satisfies the 2 following conditions for all $x \subset \mathcal{D}$:*

1. *$\lambda_x \leq \alpha_x \times \frac{\epsilon}{\Delta_{x,x'}}$ for all $x' \sim x$;*
2. *$|1 - \frac{\lambda_{x'}}{\lambda_x}| \leq (1 - \alpha_x) \times \frac{\epsilon}{ln(1/\delta)}$ for all $x' \sim x$,*

*and for some values $0 \leq \alpha_x \leq 1$. Then $\mathcal{A}$ is $(\epsilon, \delta)$-differentially private.*

**Remark 9** • One can easily derive analog results for noises with respect to other distributions, such as instance-based Gaussian noise.
• Condition 1 in Theorem 8 is equivalent to the condition

$$\lambda_x \leq \alpha_x \times \frac{\epsilon}{LS(x)} \ ,$$

illustrating that the amplitude $\lambda_x$ is calibrated to local sensitivity.
• Condition 2 requires that amplitudes of 2 neighbor instances $\lambda_x$ and $\lambda_{x'}$ should be close one from each other when $x \sim x'$, where the distance between them is measured by the privacy parameters $\epsilon$ and $\delta$.

## 3.2 LLS, the Largest Local Sensitivities mechanism

Since the noise error of the mechanism is precisely $\frac{1}{\lambda_x}$, our goal is to design an algorithm that chooses values of $\lambda_x$ as large as possible, while satisfying both conditions in Theorem 8. This obviously permits to design mechanisms with less noise, while keeping strong guarantees of privacy. In this section, we design an algorithm that achieves these conditions.
Let $f : \mathcal{D} \to R$ be a query and consider values $(LS(x))_{x \subset \mathcal{D}^n}$, local sensitivities of $f$ that can be ordered increasingly as follows: $(LS_1, ...LS_r)$, $LS_i \leq LS_{i+1}$. Notice that with such notations, we have $LS_r = GS$. We will denote

$$D_k = \{ \ x \mid LS(x) = LS_k \ \},$$

that we call the $k$-th level of sensitivities. Moreover, we will make use of the notation $k \sim l$ when there exist $x \sim x'$ such that $x \in D_k$ and $x' \in D_l$. We also write $x < x'$ (resp. $x \leq x'$) if $x \in D_k$, $x' \in D_l$ for some $k < l$ (resp. $k \leq l$).

Now we explain the design of our algorithm LLS. Ideally, an instance-based noise algorithm would have a noise amplitude at level $k$ equal to $\lambda_k = \frac{\epsilon}{LS_k}$. Unfortunately, it is possible that condition 2 in Theorem 8 is not satisfied, reflecting the fact that local sensitivity is not private in general. We will construct

an algorithm that results in a mechanism satisfying both conditions in Theorem 8 for $\alpha_x = 1/2, x \in \mathcal{D}$. Hence our goal is to find values $(\lambda_k)_k$ as large as possible such that $\lambda_k \leq \frac{1}{2} \times \frac{\epsilon}{\Delta_k}$ and condition 2 hold.
A straightforward computation results in the following proposition, which drives our choice of parameters $(\lambda_k)_k$ in algorithm LLS.

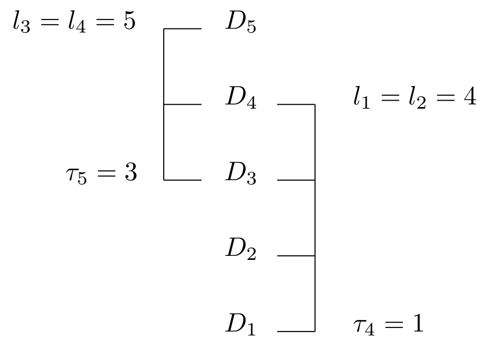
**Proposition 10** *Let $0 < LS_k < LS_{k+1}$, and $0 < t < 1$. Assume that the following relations hold:*

$$\lambda_{k+1} \leq \lambda_k \leq \min( \frac{\epsilon}{2 \times LS_k} \ , \ \lambda_{k+1} \times (1 + \frac{1}{2}t) \ ). \quad (*)$$

*Then we have*

$$|1 - \frac{\lambda_k}{\lambda_{k+1}}| \leq \frac{1}{2} \times t \ ,$$
$$|1 - \frac{\lambda_{k+1}}{\lambda_k}| \leq \frac{1}{2} \times t \ .$$

We warn the reader that two neighbor datasets $x \sim x'$ do not always belong to consecutive levels $D_k, D_{k-1}$: in other words, the situation $x \in D_k$ and $x' \in D_l$, $l < k - 1$ can occur. For that reason, we will consider in the sequel $\tau_k$, defined as the lowest neighbor level to $D_k$.
So that condition 2 in Theorem 8 holds, the noise at some level $D_k$ requires to be calibrated to the highest neighbor level, whose index is denoted by $l_k$.
An illustration of values of $\tau_k, l_k$ on some example is given in the following picture.



*We draw a link between two levels $D_k$ and $D_{k'}$ when there exist two neighbors $x$ and $x'$ such that $x \in D_k$ and $x' \in D_{k'}$.*

The heuristic for our algorithm goes as follows. We build the sequence $(\lambda_k)_k$ by descending induction, starting from $\lambda_r = \frac{\epsilon}{2LS_r}$. As is clear in Figure 1, our strategy allows the sanitizer to compute only the local sensitivities $LS_k$ larger

than $LS(x)$, when answering the query $f(x)$.
Here are explanations on how to choose $\lambda_k$ from $(\lambda_l)_{l\geq k+1}$. Given $\lambda_{k+1}$ ($\leq \frac{\epsilon}{2LS_{k+1}}$), we want to find $\lambda_k$ ($\leq \frac{\epsilon}{2LS_k}$) greater than $\lambda_{k+1}$, and such that the privacy conditions ($*$) hold with all previously constructed $\lambda_{k+m}$ (these conditions were kept in memory in previous steps). During step $k$, we compute the privacy conditions ($*$) between $\lambda_k$ and the lower neighbors $\lambda_{k-m}$, to be kept in memory until we reach the lowest of their levels $\tau_k = k - m_0$. Later in the algorithm at step $k - m_0 - 1$, we will be free to delete the privacy conditions associated to $\lambda_k$, in order to maximize as much as possible further values of $(\lambda_j)_{j\leq k-m_0-1}$.
In order to choose $\lambda_k \geq \lambda_{k+1}$ at step $k$, we need $LS_k$ and $LS_{k+1}$ to be not too close one from each other. A careful look at the proof shows that condition $\frac{LS_{k+1}}{LS_k} \geq (1 + \frac{1}{2} \times t)$ is required, where $t = \frac{\epsilon}{\ln(1/\delta)}$.

```
Algorithm 1: LLS Algorithm
**Input**: data set $y \subset \mathcal{D}$, query $f$,
privacy parameters $\epsilon, \delta$
**Output**: private value for $f(y)$
. set $t = \frac{\epsilon}{\ln(1/\delta)}$
. compute $LS_r$ and set $\lambda_r = \frac{\epsilon}{2LS_r}$
. **if** $y \in D_r$, **return** $f(y) + \mathrm{Lap}(\lambda_r)$
. **end if**
. **for** $l \sim r, l < r$, compute $LS_l$
. **end for**
. compute $\tau_r = \min_{l<r,l\sim r} l$
. **for** k from r-1 to $s$, with $y \in D_s$ **do**
.    **for** $l \sim k, l < k$, compute $LS_l$
.    **end for**
.    compute $\tau_k = \min_{l<k,l\sim k} l$
.    **if** $\frac{LS_{k+1}}{LS_k} < (1 + \frac{1}{2} \times t)$,
.    set $\lambda_k = \lambda_{k+1}$
.    **else** compute $l_k = \max_{\tau_l \leq k, l\sim k} l$
.    set $\lambda_k = \min(\frac{\epsilon}{2LS_k}, \lambda_{l_k} \times (1 + \frac{t}{2}))$
.    **end if**
.    **if** $y \in D_k$, **return** $f(y) + \mathrm{Lap}(\lambda_k)$
.    **end if**
. **end for**
```

Figure 1: Our instance-based mechanism LLS

Regarding privacy, we have the following property, which proof is given in appendix 8.2.

**Proposition 11** *LLS is well-defined, and is* $(\epsilon, \delta)$*-differentially private.*

# 4 Optimizations on the design of LLS

LLS mechanism achieves good performances for both noise reduction and time complexity. This section highlights these improvments, and proposes optimization and simplification to design LLS.

## 4.1 Improvments on time complexity of LLS

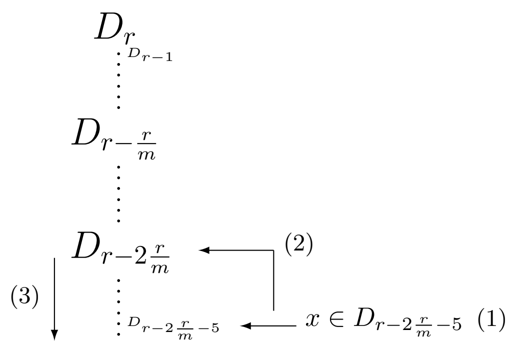
Regarding time complexity, the main advantage in LLS is that it requires only the computation of the largest local sensitivities. This is in sharp contrast with the Smooth Sensitivity technique from [16]. Indeed, the Smooth Sensitivity $S^*_{f,\beta}(x)$ requires the knowledge of *all* local sensitivities, in order to answer privately a single query $f(x)$. This advantage on execution efficiency will be illustrated through experiments in Section 6.
In a situation where multiple queries $f(x), x \subset \mathcal{D}$ are asked, and where only datasets from levels $D_k$ with small $k$ are more likely to be asked, it is not necessary to repeat the entire *descent* loop in Algorithm 1. Instead, a better strategy would consist in keeping track of some well-chosen tuples $(k, \lambda_k, D_k)$, in order to avoid computations that were already processed. In practice, the server which performs sanitization can define a storage budget, that we denote by $m$: the server decides to store at most $m$ tuples $(k, \lambda_k, D_k)$. For simplicity in exposition, we assume $r$ to be an integer multiple of $m$ in the sequel. Then the sequence of levels can be splitted as

$$LS_1 \leq LS_2 \leq ... \leq LS_r,$$

thats is into $m$ smaller sequences of $r/m$ consecutive values of the form

$$LS_{r-(k-1)\frac{r}{m}} \leq ... \leq LS_{r-k\frac{r}{m}},\ 0 \leq k < m.$$

Then the server keeps in memory tuples of the form $(r - k\frac{r}{m}, \lambda_{r-k\frac{r}{m}}, D_{r-k\frac{r}{m}})$ for $0 \leq k < m$, as soon as local sensitivities related to these levels are computed during the sanitization process. When a query $f(x)$, $x \in D_l$ is asked, the server looks for the largest $0 \leq k < m$ in its memory such that $l \leq r - k\frac{r}{m}$. Then instead of starting the loop in LLS at $r$, it can start at $r - k\frac{r}{m}$. As a consequence, LLS performs a descent only on databases in $\cup_l D_l$, where the union ranges over $r - (k-1)\frac{r}{m} < l \leq r - (k-1)\frac{r}{m}$. Hence, our Algorithm 1 provides a possible trade-off between storage space and time execution.
Note that the worst case for time complexity in LLS happens when *low* sensitivities are required. In particular, when no strategy is used, the very worst case in complexity for Algorithm 1 is comparable to the complexity in [16]. However, as seen previously, the sanitizer can design a strategy for more efficient computations with Algorithm 1, which may reduce considerably the time execution overhead when multiple datasets are queried.



(1) Determining the level of x
(2) Determining the closest memorized level to the one determined in step (1)
(3) Descent loop in algorithm LLS, starting from the level of step (2)

*We represent the set of levels, where $D_{r-k\times\frac{r}{m}}$ are special levels memorized on-the-fly for optimization. Algorithm LLS requires only levels below $D_{r-2\frac{r}{m}}$ on the figure, in order to answer privately query $f(x)$.*

## 4.2 *Well-ordered* databases and estimation of noise amplitude in LLS

In full generality, our intuition is that LLS gives drastically better accuracy when sufficiently many values of sensitivities $(LS_k)_k$ are far apart one from each other. On simple and rather general situations, the advantage of using LLS against a non-instance-based noise technique can be quantified theoretically. We now introduce such a situation assuming a specific configuration related to neighboring relation and sensitivities values.
Let us assume that we have $LS_1 < ... < LS_r$, and that for all $i$, a dataset $x \in D_i$ can have neighbors only in $D_{i+1}, D_i$ or $D_{i-1}$. Assume also that local sensitivities are not too close one from each other, that is $\frac{LS_{i+1}}{LS_i} \geq 1 + \frac{1}{2} \times t$ for $t = \frac{\epsilon}{\ln(1/\delta)}$. In such a configuration, we will say that the pair $(\mathcal{D}, (D_k)_k)$ is *well-ordered* with respect to the query $f$ and the parameters $(\epsilon, \delta)$. The following proposition will allow us to compare LLS against other mechanisms in the sequel.

**Proposition 12** *Let $f : \mathcal{D} \to \mathbb{R}$ be a query, and assume that the pair $(\mathcal{D}, (D_k)_k)$ is well-ordered with respect to $(f, (\epsilon, \delta))$. Then to answer a query $f(x)$, $x \in D_k$, parameter $\lambda_k$ required in LLS is given by the formula*

$$\lambda_k = \frac{\epsilon}{2LS_r} \times (1 + \frac{1}{2}t)^{r-k} \text{ , } \textit{where } t = \frac{\epsilon}{ln(1/\delta)}.$$

## 4.3 When is it interesting to apply instance-based noise mechanisms ?

A careful look at the estimates of the previous section on particular cases leads to an interesting phenomenon: if privacy parameters $\epsilon, \delta$ are chosen too small, then there is no point in using instance-based noise mechanisms instead of standard Laplacian mechanism. For concrete applications, the sanitizer is then in a trade-off situation where instance-based noise mechanisms are interesting options only for a particular range of the privacy parameters. We show that our estimates may help the sanitizer to choose whether or not it is worth using such techniques.

In all the current section, we assume that the pair $(\mathcal{D}, (D_k)_k)$ is well-ordered for $(f, (\epsilon, \delta))$. As we will be interested in the behaviors of privacy parameters for small values, it is worth noting that $(\mathcal{D}, (D_k)_k)$ is also well-ordered for $(f, (\epsilon', \delta'))$ for any smaller values $\epsilon' \leq \epsilon$ and $\delta' \leq \delta$. We chose to compare accuracy of Laplacian mechanism to that of other mechanisms by measuring the $\ell_1$-mean error, which is given by the following formula

$$\mathrm{err}_{\mathcal{A}} = \mathbb{E}(|Z_x|)$$

for a sanitization randomized mechanism $\mathcal{A}$ of the form $\mathcal{A}(x) = f(x) + Z_x$. For standard Laplacian noise, this error is equal to $\mathrm{err}_{Lap} = \frac{LS_r}{\epsilon}$ ($LS_r = GS$ is the global sensitivity with our notations).

Let $t = \frac{\epsilon}{\ln(1/\delta)}$. By Proposition 12, we have

$$\mathrm{err}_{LLS} = \frac{2 \times LS_r}{\epsilon} \times (1 + \frac{t}{2})^{k-r}$$

whenever the queried dataset $x$ is in level $D_k$, that is $LS(x) = LS_k$. Define $l = r - k$. For small values of $t$ (that is for a good level of privacy), we have the following estimations

$$\begin{aligned} \mathrm{err}_{LLS} &= \frac{2 \times LS_r}{\epsilon} \times \exp(-l \times \ln(1 + \frac{t}{2})) \\ &\sim_{t<<1} \frac{2 \times LS_r}{\epsilon} \times \exp(-\frac{lt}{2}). \end{aligned}$$

Then it is clear that for too small values of $lt$, mechanism $LLS$ is useless compared to Laplacian mechanism. At the opposite, a reasonable trade-off can be found between sufficiently large values of $l$ (low sensitivities) and not too small values of $t$ (reasonable amount of noise). To help the reader figure out the advantageous trade-offs, we display the graph in Figure 2. The graph reflects a natural and interesting trade-off: if privacy parameters go smaller (small values of $t$), only *low* levels (that is large values of $l$) may find benefits from our instance-based noise mechanism.

As will be shown in our experimental Section 6, a similar behavior can be observed for other variants of instance-based noise mechanisms: too small values of privacy parameters cancel the benefits of exploiting local sensitivities. In

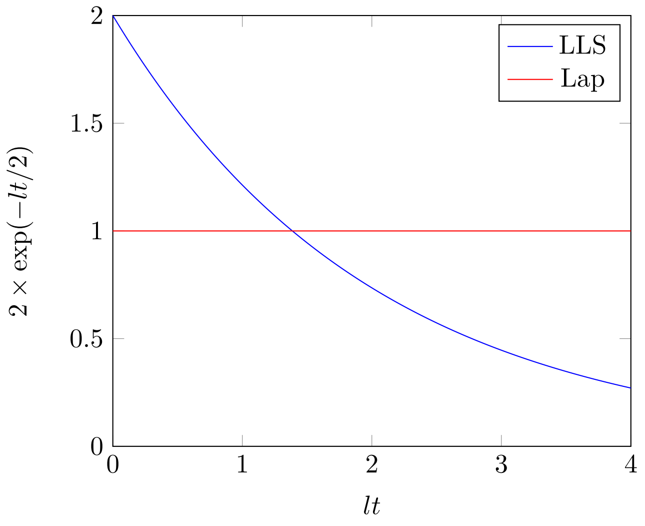


Figure 2: Relative comparison between LLS and Laplacian amplitude noises for small values of $t = \frac{\epsilon}{\ln(1/\delta)}$

the case of a Smooth Sensitivity $S^{\star}_{f,\beta}(x)$, this phenomenon is indeed a natural consequence of the convergence $S^{\star}_{f,\beta}(x) \to GS$ as $\beta = t/2 \to 0$ (see Section 5.2 for details on $S^{\star}_{f,\beta}(x)$).

## 5 Theorical evaluations

### 5.1 Genericity, accuracy and efficiency of LLS mechanism

The main remarkable features of our algorithm are summarized in the list below:

1. **Full generality on queries:** To perform well, the only assumption our algorithm requires is that local sensitivities are not too close one from each other. Anyway, this assumption is obviously necessary for *any* instance-based noise mechanism aiming at providing more accurate results than mechanisms calibrated to global sensitivity. This *gap in sensitivities* is the only assumption about the query and the database considered in this paper.

2. **An appealing trade-off between time complexity and accuracy:** Compared to concurrent instance-based noise techniques, LLS always provides a solution which appears among both the most efficient and the most accurate mechanisms. This contrasts with prior works that satisfied good performances for only one of these two aspects. Moreover, as explained

in Section 4.1, time efficiency in LLS can be optimized on demand, to the price of a small loss in the storage space.

3. **Computation of a theoretical bound on the error:** In many situations such as for well-ordered databases (see Section 4.2), we are in position to provide a theoretical estimate on the average error in algorithm LLS. This is important to decide if it is worth using such an instance-based noise mechanism or not, and also to tune the privacy parameters adequately.

## 5.2 Instance-based noise mechanisms from prior works

Here we recall some facts about concurrent mechanisms to our.

***Smooth Sensitivity SS from* [16]**

The $\beta$-smooth sensitivity $S^{\star}_{f,\beta}(x)$ of a query $f$ at $x \subset \mathcal{D}$, is given by the formula:

$$S^{\star}_{f,\beta}(x) = \max_{y \subset \mathcal{D}} \left( LS(y) \cdot \mathrm{e}^{-\beta d(x,y)} \right).$$

Using notations from Section 3.1, an $(\epsilon, \delta)$-differentially private mechanism is obtained for the values $\lambda_x = \frac{\epsilon}{2S^{\star}_{f,\beta}(x)}$, and $\beta = \frac{\epsilon}{2\ln(1/\delta)}$. This was stated in [16]. A simple proof of this privacy result using our theorem 8 is given in the Appendix 8. Now we highlight the important benchmarks between Smooth Sensitivity technique and LLS.

• LLS is efficiently computable whenever local sensitivities are. This is in sharp contrast with Smooth Sensitivity technique from [16], since the computation of Smooth Sensitivity has been proved to be NP-hard in some cases.
• Even when Smooth Sensitivity is efficiently computable, LLS has much better time complexity. Indeed, LLS requires only the computation of the largest $r-k$ local sensitivities to answer a query $f(x)$, $x \in D_k$. Smooth Sensitivity requires an exhaustive list of values $LS(x)$, for the computation of a single $f(x)$, $x \subset \mathcal{D}$.

**Example 13** Let us consider the example of the median $f_{med}$, which is detailed in Section 3.1 [16]. On an instance where the database is given by $n$ points $x_i = \Lambda i/n$, Smooth Sensitivity technique requires a noise amplitude equal to $\Lambda/\epsilon n$ to reach $(\epsilon, \delta)$-differential privacy. From Claim 3.3 [16], this can be done in time $O(n^2)$ (in time $O(n)$ if one uses an approximation of Smooth Sensitivity $S^{*}_{f,\beta}$ for $f = f_{med}$).
Let us fix values of parameters $\epsilon, \delta$ such that $\frac{1}{n} \geq \frac{t}{2}$, and let $x \in D_k$. By virtue of Proposition 12, LLS has noise amplitude $\frac{2\Lambda}{n\epsilon} \times (\frac{1}{1+\frac{1}{2}t})^{n-k}$, and performs in time $O(k)$.
Errors due to noise addition in LLS and SS mechanisms are comparable for small values of $t$, that is for a *good* level of privacy. However, time execution is clearly better for LLS.

**Example 14** Well-ordered databases can be found in many simple and realistic statistical tasks. For instance, consider the database $\mathcal{D} = \{x_1, ...x_n\}$ of employees salaries $0 \leq x_1 \leq x_2 \leq ... \leq x_n$, and using the same neighboring relation as in Example 5. Assume that we consider query $f(x) = x_1 + x_2 + ... + x_k$ for some $k$, that is the sum of the $k$ lowest salaries in the company. With notations as above, $r$ is the number of distinct salaries in the company, and $LS_1 < ... < LS_r$ is the ordered sequence of distinct salaries in the company.
When applying algorithm LLS over $m = kr$ queries, the amplitude of noise required is of the order $O(\frac{k\ln(r)}{r})$ when all levels $D_i$ are queried $k$ times each.
When exclusively *low* salaries are queried, we have an amplitude of noise of the order $O(\alpha^r)$ for some parameter $0 < \alpha < 1$ depending only on $\epsilon, \delta$. By contrast, standard Laplacian technique would give an amplitude of the order $O(1)$.

***Noisy Local Sensitivities NLS from [11]***

In [11], it is shown that computing Smooth Sensitivity of some statistics on graphs is a NP hard problem. Authors of [11] focus on differentially private graphs statistics in the sense of edge privacy, that is two graphs are considered neighbors if they differ one from each other by exacty one edge. Let $f_{2\Delta}$ the number of 2-triangles (a 2-triangle is given by two triangles sharing a common edge). Then it is NP hard to compute $S^*_{f,\beta}$ for $f = f_{2\Delta}$ (see Theorem A.1 in [11]).
The main purpose in [11] is to provide an alternative instance-based noise method to Smooth Sensitivity, called Noisy Local Sensitivities (NLS). In NLS, the idea consists in computing private versions of local sensitivities. This is done by means of a second order local sensitivity, which is a sensitivity of local sensitivities, when considering the latter as functions over $\mathcal{D}$. Their algorithms apply only on graph statistics, since they use specific upper bounds on local sensitivities of statistics in graphs.

• Results in [11] give reasonable accuracy only if one wants to release privately $f_{2\Delta}(G)$ for some sufficiently connected graph $G$. The precise assumption made by the authors is the existence of a pair of vertices in $G$ which have a number of common neighbors significantly greater than $\frac{\ln(1/\delta)}{\epsilon}$. Our algorithm does only make some mild assumption on the graphs we want to release, and in particular can be used even when graphs do not have many connections. More precisely, since we are looking for values of $t$ which are not too large, the condition $\frac{\overline{LS}_{k+1}}{\overline{LS}_k} \geq 1 + \frac{1}{2}t$ should hold in most cases, and especially when the values $a_H$ are not too large (this is in contrast with the assumption of [11] where it is required $a_H$ to be much larger than $1/t$).
• For the first $f(x)$ $(x \in D_l)$ query asked, computation of the sequence $(\overline{LS}_k)_{k \geq l}$ is required, and this can be computed in time $O(d_{max}m^2)$ where $m$ is the number of possible edges in the overall graph $G$ considered, and $d_{max}$ a bound on the degree of vertices. For next computations, some elements of the list $(\overline{LS}_k)_{k \geq l}$ already computed can be used, and many less computations are required (see section 4.1). The computation time is more likely to be $O(d_{max}m)$ after a few

queries.
• As pointed out in [18], Noisy Local Sensitivities method applied to $f_{2\triangle}$ holds only for a restricted set of the privacy parameters $(\epsilon, \delta)$, namely for $\epsilon \in (0, 0.6)$. LLS does not suffer from such a restriction, which can be crucial in practice. Indeed, some more sensitive data require higher values of $\epsilon$ in the differentially private mechanism in order to keep some utility.

***Ladder Functions LF from [18]***

Other approaches were used to answer queries on graphs in a differentially private manner (e.g. [2], [4], [18]). Among recent studies, one of the most successful seems to be the one using *ladder functions* in [18]. In a nutshell, ladder functions method performs a refined version of the well-known exponential mechanism (see [14]), where the amplitude noise $\lambda_x$ to answer a query $f(x)$ is calibrated with respect to some ladder function $I_t(x)$. Functions $I_t(x)$ play the same role as local sensitivity in our framework, and allows for a favorable instance-dependent result. For the case of counting queries on graphs, Ladder Function mechanism uses a system of convergent ladders, that permits to sample from exponential mechanism efficiently (which is a priori a non-efficient task).

• Time complexity of ladder functions is similar to that of Smooth Sensitivity (see section 6.1 in [18]). We have already seen that LLS was designed with the purpose to have drastically better time complexity than Smooth Sensitivity algorithm. This is obviously the case when asking a large number of private queries, as highlithed in Section 4.1 ). This makes LLS a first-line option among instance-based noise mechanisms regarding time performances.
• Methods from [18] were designed exclusively for subgraph countings. It is not clear how to adapt the ladder functions framework for general queries, such as Example 14 or Example 13. By contrast, LLS can be used to answer any type of queries.

## 5.3 Comparison on a small graph

In order to understand the way how each instance-based noise algorithm proceeds, and to better understand the main differences between these methods, we present a simple sanitization task: releasing a differentially private version of $f_{2\triangle}$ using mechanisms SS, NLS, LF and LLS.

Our simple example is defined as follows. Many notations and some results are retrieved from [11], which first gave an in-depth study of differential privacy on query $f_{2\triangle}$. Consider a based (undirected) graph $G_0$ with 6 nodes such that any pair of nodes is connected by an edge. We allow to apply query $f_{2\triangle}$ on the dataspace $\mathcal{D} = \mathcal{G}_6$, the set of all subgraphs of $G_0$. The privacy model we consider here is edge privacy.

Authors from [11] introduce a particularly convenient upper bound on local sensitivity. This estimate allows for much easier computations and provable privacy. We will use here the same upper bound, as an estimate of local sensitivity for each of mechanisms SS, NLS, LF and LLS. For a subgraph $H \in \mathcal{G}_6$, and an edge $e \notin H$, we denote by $H + e$ the graph obtained from $H$ by adding the edge $e$. For an edge $e \in G$ and $H \in \mathcal{G}_6$, we use the notation $a_e^H$ for the number of triangles in $H$ involving edge $e$. Set $a_H = \max_{e \in H+e', e' \notin H} a_e^{H+e'}$, and $\overline{LS}(H) = \frac{5}{2} \times a_H$. By computations from [11], $\overline{LS}$ is an upper bound approximation on $LS$.

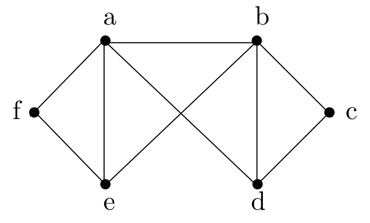
It appears that upper bounds $\overline{LS}$ have also an additional interesting feature for our applications.

**Proposition 15** *Let $H \sim H', H, H' \in \mathcal{G}_6$. Then we have*

$$|\overline{LS}(H) - \overline{LS}(H')| \in \{0, 5/2\}.$$

Note that such a configuration for sensitivities is attractive to apply LLS mechanism. Defining sensitivities levels $D_k$ with $\overline{LS_k}$ instead of $LS_k$, the pair $(\mathcal{G}_6, (D_k))$ is well-ordered with respect to $(f_{2\Delta}, (\epsilon, \delta))$, for sufficiently small values of privacy parameters $\epsilon, \delta$.

Now our goal is to sanitize the query result $f_{2\Delta}(G)$ for the graph $G \in \mathcal{G}_6$ defined below. This graph was also used in [18] as a simple example for explanations.



We fix privacy parameters values $\epsilon = 5$ and $\delta = 1/2000$. This choice may seem unreasonable regarding Definition 6, since $\epsilon = 5$ would not impose sufficiently tight constraints on *neighbor* distributions. Indeed, we chose these values to have at least meaningful results for (some of) the concurrent mechanisms. Then a *large* value for $\epsilon$ is not surprising: we consider a small graph to ease the explanations, which results in a difficult sanitization objective. Much smaller values of $\epsilon$ will be used for real applications in Section 6, for larger graphs closer to real-life applications of privacy-preserving data mining.

***Smooth Sensitivity SS***

To estimate the $\beta$-smooth sensitivity of $f_{2\Delta}$ at $G$, we compute

$$s = \max_{G' \in \mathcal{G}_6} \left( \overline{LS}(G') \cdot \mathrm{e}^{-\beta \mathrm{d}(G,G')} \right).$$

The previous computation of maximum does not require to consider graphs having less edges than $G$. There are $\binom{6}{k}$ graphs in $\mathcal{G}_6$ with $k$ edges greater than $G$. Estimation $s$ is obtained up to the computation of $\sum_{k=1}^{6}\binom{6}{k} = 64$ local sensitivities. Performing this, we obtain $s = \max(15/2, 10e^{-\beta})$ where $\beta = \frac{\epsilon}{2\ln(1/\delta)} \approx 0.33$. Finally, Smooth Sensitivity adds a noise with amplitude $\frac{1}{\lambda_{SS}} = \frac{2\max(15/2, 10e^{-\beta})}{\epsilon} = 3$.

***Noisy Local Sensitivity NLS***

Let us turn now to our instantiation of Algorithm 1 from [11] to compute Noisy Local Sensitivity on $f_{2\Delta}(G)$. We use here notations as in [11]. We set $\epsilon' = \frac{\epsilon}{3}$, $\delta' = \frac{\delta}{3}$ and $a_{max} = 1$. Then the computation of the private local sensitivity is as follows:

$$\widetilde{a}_{max} = 2 + \text{Lap}\left(\frac{1}{\epsilon'}\right) + \frac{\ln(1/\delta')}{\epsilon'}$$
$$\widetilde{LS} = 15/2 + \text{Lap}\left(\frac{4\widetilde{a}_{max}}{\epsilon'}\right) + \ln(\frac{1}{\delta'}) \cdot \frac{4\widetilde{a}_{max}}{\epsilon'}.$$

As a result, NLS returns a noise amplitude equal to $\frac{1}{\lambda_{NLS}} = \frac{\widetilde{LS}}{\epsilon'} \approx 72$.

***Ladder Functions LF***

To compute the noise of $f_{2\Delta}$ using LF, we follow the algorithm NoiseSample in [18]. We compute

$$I_t(G) = \min\left(\frac{15}{2} + \sum_{i=0}^{t-1} 8 + 4i, GS(f_{2\Delta})\right)$$

for $t \in \{1, 2, 3\}$ and we take $C = GS(f_{2\Delta}) = 10$. After computing the weight for each range, we see that we have more than 50% to return $\frac{1}{\lambda_{LF}} = 6.75$ and more than 40% to return $\frac{1}{\lambda_{LF}} = 0$.

***Largest Local Sensitivities LLS***

Using the upper bound approximation $\overline{LS}$, the data is well-ordered. Two consecutive values of sensitivities $\overline{LS}(H) \in D_k$ and $\overline{LS}(H') \in D_{k+1}$ are separated by a gap equal to 0 or 5/2. So in algorithm LLS, we have $\tau_k = k - 1$, $l_k = k + 1$ and $\overline{LS}_i = i \cdot 5/2$ for $i \in \{1, 2, 3, 4\}$. All stages of Algorithm 1 are detailed in Appendix 8. The algorithm requires an amplitude noise equal to $\frac{1}{\lambda_{LLS}} = \frac{1}{\lambda_3} = 3$.

***Comparisons***

The results obtained previously can be compared easily. This section does not aim to give a general comparison between the different instance-based methods, but rather to understand some important differences between the mechanisms on a simple example. The *intuitions* which we highlight below will be confirmed on real datasets evaluations in the next section.

We introduce a few notations for *high-level* comparisons. We will write $A \prec_u B$ (resp. $A \prec_t$) if method $B$ performs much better than method $A$ regarding utility (resp. time complexity). Symbol $A \preceq_u B$ (or $A \preceq_t B$) means that $B$ performs better than $A$, but that they are *close* to provide equivalent results. Note that for the above methods, utility is measured by the noise amplitude added by the mechanism, for a same level of privacy.

| | |
|---|---|
| **Utility** | $NLS \prec_u LF \prec_u LLS \approx_u SS$ |
| **Efficiency** | $SS \prec_t LF \preceq_t NLS \approx_t LLS$ |

The comparisons displayed in the table are not surprising, and reflects the natural features of the different approaches:

- NLS performs poorly for such privacy parameters, since there are multiple sources of noise and several amplifications of the error whenever a single privacy parameter is chosen smaller. SS, LF, and LLS provide comparable results. Note that it was expected that SS method would result in one of the best options regarding accuracy of the mechanism, since it calibrates the noise to the best smooth upper bound (see Appendix 8 for more details).

- SS requires the computation of a large number of sensitivities. At the opposite, the three other techniques are based on a single sensitivity value. Even if this task is negligible on such a basic example, it is clear that the sampling step is less efficient for the LF case than for NLS or LLS, because of the inherent complexity of exponential mechanism noise sampling.

## 6 Experimental Evaluations

To support the idea that LLS mechanism should be considered as an interesting option to answer queries privately, we realized experimental evaluations in which we compare LLS to the current state-of-the-art mechanisms. All the algorithms were implemented in Scala on a personal computer with 2.30 GHz Intel i5 CPU and 8 GB RAM Memory.

### 6.1 Experiments settings

We use two real-life datasets to conduct our experiments. The first one is a dataset that contains the average annual salary of 148654 employees from San Francisco. The second dataset is Gowalla data, which contains 6,442,890 check-in locations of 196,586 users over the period of Feb. 2009 to Oct. 2010. Before applying our queries, we built gaphs from the dataset Friendship network of Gowalla users, described as follows: each node represents a user, and an edge is drawn between two nodes $a$ and $b$ when the two users corresponding to $a, b$ know each other. Datasets are respectively referred as *Salaries* and *Gowalla* in the sequel.

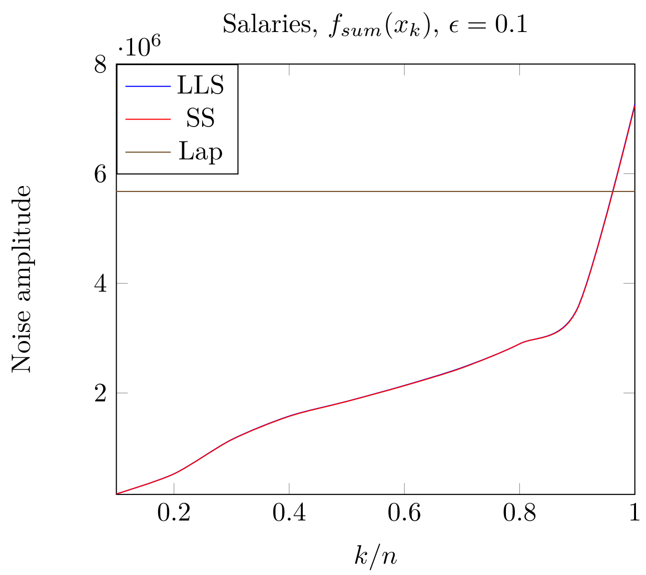


Figure 3: The private sum of the $k$ smallest salaries, among $n = 148654$ salaries.

On Salaries dataset, we compare SS and LLS by applying the mechanisms on the query $f_{sum}$, introduced in Example 5. Recall that $f_{sum}(x_k)$ is a notation for the sum of the $k$ smallest salaries in the dataset. On Gowalla dataset, we extract subgraphs $H$ of the Gowalla based graph $G_0$. Then we applied the following sanitization mechanisms to some statistics on subgraphs $H \subset G_0$: *LLS*, LF, NLS and Laplacian mechanism (referred as *Lap*). These mechanisms are compared on sanitization of two graph queries: $f_{edge}$ counting the number of edges in the queried graph, and $f_{2\Delta}$ counting the number of 2-triangles.

Note that for (Salaries, $f_{sum}$) and (Gowalla, $f_{edge}$), mechanisms LF, NLS are not displayed. Indeed, the latter methods were designed only for particular graph queries and do not apply to these cases. At the opposite, results concerning SS mechanisms on graph queries are not shown in our figures, since our experiments confirm the well-known fact that time execution of SS mechanism increases too much with the size of the queried graph.

In all the section, parameter $n$ will refer to the size of the database, that is the number of salaries in Salaries, and the number of nodes for Gowalla base graph $G_0$. The size (number of nodes) of a subgraph $H \subset G_0$ will be denoted by $n_H$. Whenever a privacy parameter $\delta$ will be used together with $\epsilon$, we will take $\delta = \epsilon/2n$. Parameter $\epsilon$ will be precised on figures, and may vary depending on the conducted experiment.

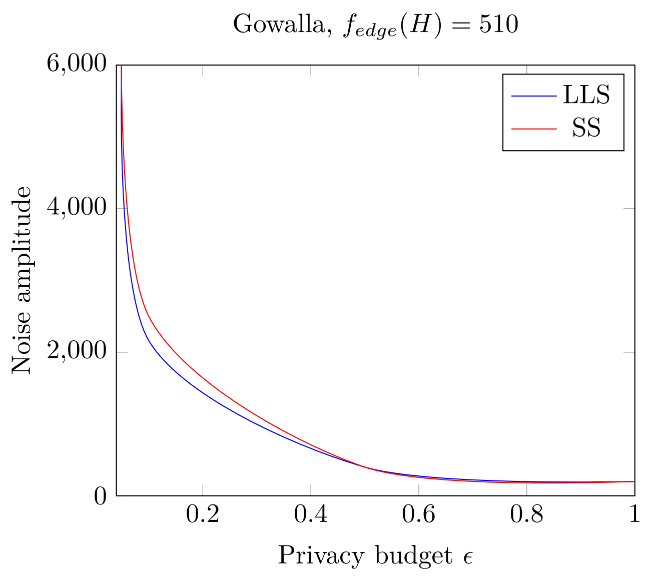


Figure 4: Dependence of noise amplitude on privacy parameter $\epsilon$ (with $\epsilon = \delta/2n$), for a graph $H$ of size $n_H$

## 6.2 Analysis

Figure 3 illustrates how close mechanisms LLS and SS are, regarding noise addition required to sanitize *basic* queries. This similarity is not surprising: LLS is in some sense a *truncated* version of SS mechanism, for which only the largest local sensitivities are used, and not all local sensitivities. This *truncation* has the natural effect to provide a very fast execution (less than 130 ms for every $k$), while SS would have a long execution time even for small values of $k$ (more than three minutes for $k/n < 0.6$).

We also applied sanitization mechanisms LLS and SS on $f_{edge}$ and some Gowalla subgraph $H \subset G_0$ of size $n_H = 100$. According to Figure 4, LLS gives slightly better results for small values of $\epsilon$. Note that this is not relevant, since the amplitude noise is too large compared to the real value of the query output $f_{edge}(H)$. A more interesting fact, which confirms a point already discussed in the paper, is the similarity of amplitude noise for both algorithms and for reasonable values of privacy parameters. As in the case of aggregation of salaries, execution time of LLS is faster than that of SS. We have applied these mechanisms on larger Gowalla subgraphs: the noise amplitudes remain similar for the two algorithms, but the difference in execution time significantly increases in favor of our algorithm.

Now we turn to private releases of $f_{2\Delta}$ on Gowalla subgraphs $H \subset G_0$ using the approximation $\overline{LS}(H)$ defined and detailed in the previous section. It appears that mechanism SS has nearly the same noise amplitude (for small values

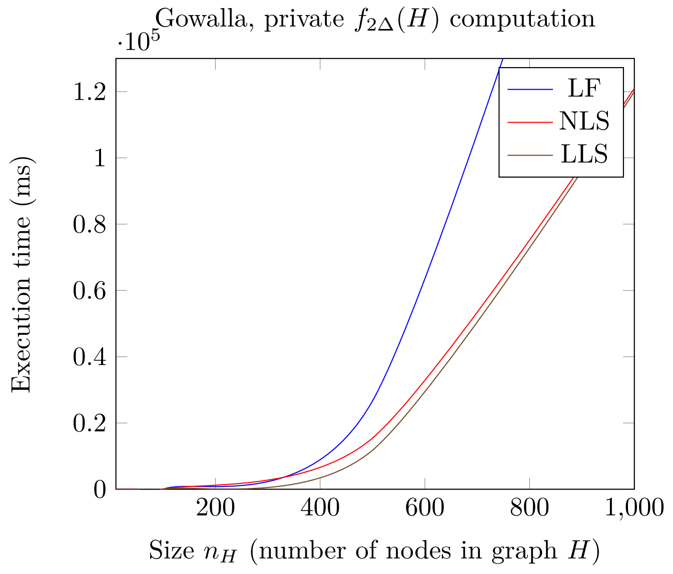


Figure 5: Average time execution for graphs $H$ of size $n_H$.

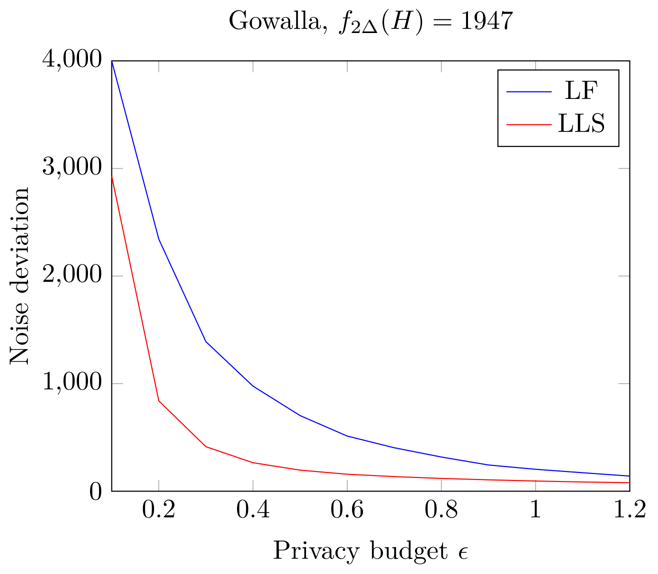


Figure 6: Accuracy for 2-triangles counting, on a graph $H \subset G_0$ with $n_H = 100$ nodes and $H \in D_{19}$

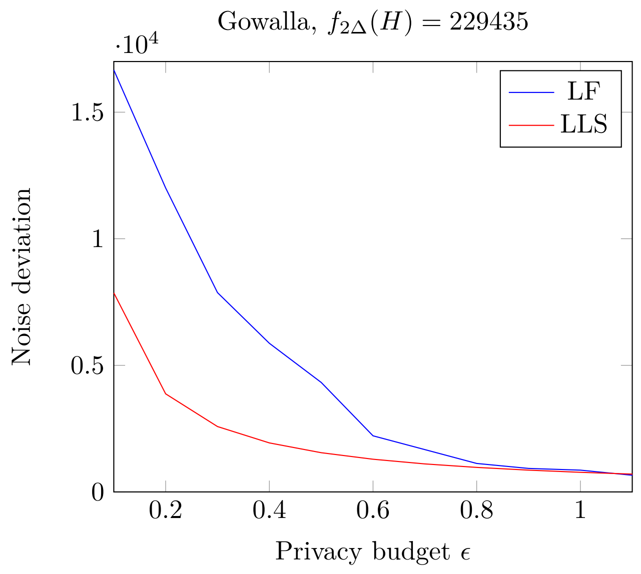


Figure 7: Accuracy for 2-triangles counting, with a graph $H \subset G_0$ with $n_H = 500$ nodes and $H \in D_{155}$

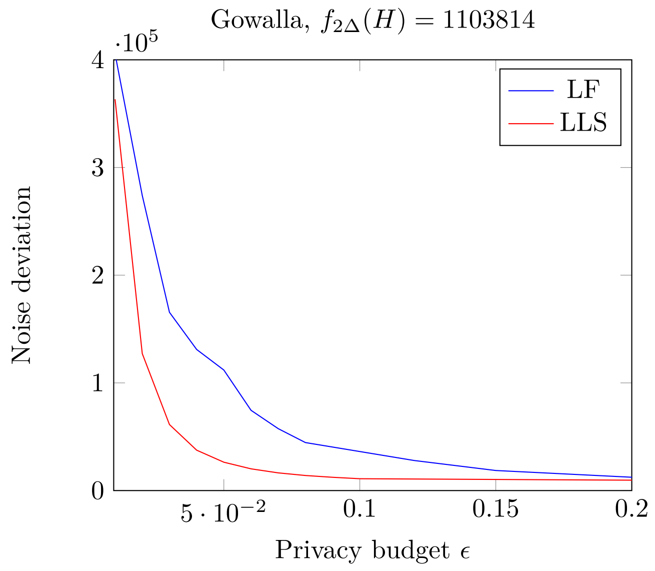


Figure 8: Accuracy for 2-triangles counting, with a graph $H \subset G_0$ with $n_H = 1000$ nodes and $H \in D_{222}$

of the graphs size $n_H$) as LLS, but its time execution increases exponentially and the mechanism can not be applied even for medium values of $n_H$. It was expected since computing Smooth Sensitivity is NP Hard in the case of $f_{2\Delta}$. At the opposite, NLS performances are very fast but its noise amplitude increases unreasonably. This fact is certainly due to the requirement of several sources of noise in NLS mechanism (for further details, see the formulae in Section 5.2).

For clarity in our figures, we used LF mechanism as a reference point for our comparisons. Indeed, other instance-based noise mechanisms have been extensively compared to LF mechanism in [18], and experiments from that prior work showed that LF was the most challenging mechanism to be compared with. To compare the different mechanisms, we compute the noise deviation, that is the average distance between the real output of the query and its private version. For instance, noise deviation is calibrated to the noise amplitude $1/\lambda_{LLS}$ for LLS mechanism. For LF mechanism, the mechanism was applied 100 times on the subgraph $H \subset G_0$ and the noise deviation is defined as the mean of the noises added to the real output $f_{2\Delta}(H)$.

We compare instance based-noise mechanisms LLS and LF for reasonably small values of $\epsilon$ ($\epsilon < 1.2$), and for subgraphs $H$ picked at different *levels* of sensitivity ($D_{19}, D_{155}, D_{212}$). Such privacy values correspond to a meaningful privacy guarantee in Definition 6. It is clear that for all configurations in Figures 6, 7, 8, there is a range $[0, \epsilon_0]$ in which $\epsilon$ induces a significantly better result for LLS than for LF. Outside this range, both mechanisms are comparable regarding noise perturbation. We can also notice that the critical value $\epsilon_0$ for which the two mechanisms errors *converge* depends on the queried level $D_k$ ($H \in D_k$). The smaller the level $D_k$, the larger the critical value $\epsilon_0$ is. This confirms that our mechanism LLS particularly well apply to *low sensitivities levels*, and *deep levels* datasets.

To summarize, it appears that for simple queries (mean, sum, median), SS and LLS both can be considered as interesting options. But for more involved queries ($k$-triangles counting, edges counting), our experiments would suggest to consider LLS as a first choice, since it seems to provide the best trade-off between time execution and accuracy, for any possible range of the privacy parameter $\epsilon$. The table below summarizes the different comparison relationships obtained from our experiments (with notations from the end of Section 5.2).

| **Query** | **Utility** | **Efficiency** |
|---|---|---|
| $f_{sum}$ | LLS$\preceq_u$ SS | SS $\preceq_t$ LLS |
| $f_{edge}$ | SS $\preceq_u$ LLS | SS $\prec_t$ LLS |
| $f_{2\Delta}$ | NLS$\prec_u$ LF $\preceq_u$ LLS | LF$\prec_t$ NLS $\preceq_t$LLS |

## 7 Conclusion and future work

In this paper, we introduced novel techniques to design differentially private instance-based noise mechanisms. Our approach exploits the standard idea of instance-based noise mechanisms to optimize the amount of noise required for differential privacy. The novelty of our methods rely in the use of a restricted number of local sensitivities, which results in drastic optimizations regarding time execution of the sanitization mechanism. We instantiated these ideas in a novel differentially private mechanism, that we called Largest Local Sensitivities mechanism (LLS). We compared LLS mechanism to state-of-the-art ones, and we showed that LLS should be considered as a first-order option in many cases, including basic queries (means, medians) and more involved graphs statistics (2-triangles countings).

We believe that our Theorem 8 has a wide potential for further applications, because it is by nature generic and easy to handle. For instance, it would be interesting to investigate analogs of instance-based noise mechanisms in the context of geo-indistinguishability [1] [3], a variation on differential privacy designed for location privacy at the user scale.

As another future direction of research, we would like to consider more carefully the case of multi-dimensional range queries. In particular, it would be certainly interesting to study in which extent the optimizations techniques from [12] could apply to the framework of our Theorem 8 and our mechanism LLS.

## References

[1] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis and C. Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. *Proceedings of the 2013 ACM SIGSAC conference on computer and communications security. ACM*, p 901-914, 2013.

[2] J. Blocki, A. Blum, A. Datta and O. Sheffet. Differentially private data analysis of social networks via restricted sensitivity. *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, p. 87-96 (2013)

[3] K. Chatzikokolakis, C. Palamidessi and M. Stronatti. Constructing elastic distinguishability metrics for location privacy. *Proceedings on Privacy Enhancing Technologies, 2015(2)*, p 156-170, 2015.

[4] S. Chen and S. Zhou. Recursive mechanism: towards node differential privacy and unrestricted joins. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, p. 653-664 (2013)

[5] C. Dwork. Differential privacy: A survey of results. *Theory and Applications of Models of Computation*, p 1-19 (2008)

[6] C. Dwork, K. Kenthapadi, F. Mc Sherry, I. Mironov and M. Naor. Our data, ourselves: Privacy via distributed noise generation. *Advances in Cryptology-EUROCRYPT*, p 486-503 (2006)

[7] C. Dwork, F. Mc Sherry, K. Nissim and A. Smith. Calibrating noise to sensitivity in private data analysis. *Theory of cryptography*, p 265-284 (2006)

[8] C. Dwork and A. Roth. The Algorithmic Foundations of Differential Privacy.

[9] S. R. Ganta, S. P. Kasiviswanathan and A. Smith. Composition attacks and auxiliary information in data privacy. *Proceedings of the 14th ACM SIGKDD international on Knowledge and data mining*, p 265-273 (2008)

[10] S. P. Kasivisiwanathan, K. Nissim, S. Raskhodnikova and A. Smith. Analyzing graphs with node differential privacy. *Theory of Cryptography, Springer Berlin Heidelberg*, p 457-476 (2013)

[11] V. Karwa, S. Raskhodnikova, A. Smith and G. Yaroslavtsev. Private Analysis of Graph Structure. *Proceedings of the VLDB Endowment*, vol. 4, no. 11, p 1146-1157 (2011)

[12] C. Li, M. Hay, V. Rastogi, G. Miklau and A. McGregor. Optimizing linear counting queries under differential privacy. *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM,* p 123-134 (2010)

[13] F. D. Mc Sherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data.* ACM, p 19-30 (2009)

[14] F. McSherry and K. Talwar. Mechanism design via differential privacy. *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium.* p.94-103 (2007)

[15] D. J. Mir and R. N. Wright. A differentially private graph estimator. *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on.* IEEE, p 122-129 (2009)

[16] K. Nissim, S. Raskhodnikova and A. Smith. Smooth Sensitivity and Sampling in Private Data Analysis. *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, p 75-84 (2007)

[17] Y. Wang and X. Wu. Preserving differential privacy in degree-correlation based graph generation. *Transactions on Data Privacy*, vol. 6, no. 2, p 127 (2013)

[18] J. Zhang, G. Cormode, C.M. Procopiuc, D. Srivastava and X. Xiao. Private release of graph statistics using ladder functions. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, p. 731-745 (2015)

# 8 Appendix

## 8.1 A proof of $(\epsilon, \delta)$-differential privacy

**Proof of Theorem 8** A sufficient condition for mechanism $\mathcal{A}$ to be $(\epsilon, \delta)$-differentially private is the existence, for all $x \in \mathcal{D}$, and for all $x' \sim x$, of subsets $E_{x,x'} \subset R$ such that $\mathbb{P}(\mathcal{A}(x) \in E^c_{x,x'}) \leq \delta$ and:

$$\frac{g_x(y)}{g_{x'}(y)} \leq e^{\epsilon} \text{ for all } y \in E_{x,x'}.$$

Note that the previous condition is equivalent to:

$$|\lambda_x |f(x) - y| - \lambda_{x'} |f(x') - y|| \leq \epsilon \text{ for all } y \in E_{x,x'}.$$

Moreover, a triangular inequality gives

$$|\lambda_x |f(x) - y| - \lambda_{x'} |f(x') - y|| \leq \\ \lambda_{x'} \Delta_{x,x'} + |\lambda_{x'} - \lambda_x| \times |f(x) - y|.$$

Hence from the assumptions, it is sufficient to find subsets $E_{x,x'}$ such that:

$$|\lambda_{x'} - \lambda_x| \times |f(x) - y| \leq (1 - \alpha_x) \times \epsilon \text{ for all } y \in E_{x,x'}.$$

Now set $E_{x,x'} = \{ \ y \mid |\lambda_{x'} - \lambda_x| \times |f(x) - y| \leq (1 - \alpha_x) \times \epsilon \ \}$. To prove the theorem, it is then sufficient to show that $\mathbb{P}(\mathcal{A}(x) \notin E_{x,x'}) \leq \delta$. For that, denote by $Z$ a Laplacian random variable of parameter $\lambda = 1$. The conclusion of the proof is a straightforward consequence of the tail bound estimation for Laplacian distribution:

$$\begin{aligned}\mathbb{P}(\mathcal{A}(x) \notin E_{x,x'}) &= \mathbb{P}(Z > \frac{(1-\alpha_x) \times \epsilon \times \lambda_x}{|\lambda_{x'} - \lambda_x|}) \\ &= e^{-((1-\alpha_x)\times\epsilon)\times\frac{1}{|1-\frac{\lambda_{x'}}{\lambda_x}|}} \\ &\leq \delta\end{aligned}$$

by assumption 2 in Theorem 8. ■

## 8.2 Privacy and consistency of algorithm LLS

We first give the proof of Proposition 10, which will simplify the proof of the main Proposition 11.

**Proof of Proposition 10** By assumption, we have

$$\begin{aligned}|1-\frac{\lambda_k}{\lambda_{k+1}}| &= \frac{\lambda_k}{\lambda_{k+1}} - 1 \\ &\leq (1+\frac{1}{2}t) - 1 \\ &\leq \frac{1}{2}t.\end{aligned}$$

From the above inequalities and the assumption $\frac{\lambda_{k+1}}{\lambda_k} \leq 1$, we obtain:

$$\begin{aligned}|1-\frac{\lambda_{k+1}}{\lambda_k}| &= \frac{\lambda_{k+1}}{\lambda_k} \times |1-\frac{\lambda_k}{\lambda_{k+1}}| \\ &\leq \frac{1}{2}t.\end{aligned}$$

■

**Proof of Proposition 11** First notice that LLS is well-defined: the induction process is valid. As assumed in this paper, $\mathcal{D}$ is connected with respect to our neighboring relation. In particular, there exists $l > k$ (when $k \neq r$) such that $\tau_l \leq k$. It follows that $l_k > k$, and then $\lambda_k$ is constructed from the sequence $(\lambda_l)_{l>k}$.
Now we need to show that the mechanism constructed in LLS satsifies condition 2 from Theorem 8 with $\alpha_x = \frac{1}{2}$, that is

$$|1-\frac{\lambda_x}{\lambda_{x'}}| \leq \frac{1}{2} \times \frac{\epsilon}{\ln(1/\delta)}$$

for all $x \in \mathcal{D}, x' \sim x$. From Proposition 10 which deals with the situation of two sensitivities (with the choice $t = \frac{\epsilon}{\ln(1/\delta)}$), we deduce that it is sufficient to prove, for all $x' \sim x$, $x' < x$, the following inequalities (referred as $(*)$ in the sequel) hold:

$$\lambda_x \leq \lambda_{x'} \leq \min(\ \lambda_x \times (1 + \frac{1}{2}t)\ ,\ \frac{\epsilon}{2 \times LS(x')}\ ).$$

Inequalities on the left in $(*)$ is simply requiring that the sequence $(\lambda_k)_k$ is decreasing. We only need to prove the monoticity in the case $\frac{LS_{k+1}}{LS_k} \geq (1 + \frac{1}{2} \times t)$. For that, we consider the following two cases:

- Case 1 : $\lambda_k = \frac{\epsilon}{2LS_k}$. Obviously, we have $\frac{\epsilon}{2LS_k} \geq \frac{\epsilon}{2LS_{k+1}}$. Moreover, we have $\lambda_{l_{k+1}} \leq \lambda_{k+1}$ (by monoticity in the previous steps), and then $\lambda_{l_{k+1}} \leq \frac{\epsilon}{2LS_{k+1}}$. Hence, in order to have $\lambda_k \geq \lambda_{k+1}$, it is sufficient that the following holds:

$$\frac{\epsilon}{2LS_k} \geq \frac{\epsilon}{2LS_{k+1}} \times (1 + \frac{1}{2} \times t).$$

  This is the case since we are in the situation where $\frac{LS_{k+1}}{LS_k} \geq (1 + \frac{1}{2} \times t)$.

- Case 2 : $\lambda_k = \lambda_{l_k} \times (1 + \frac{1}{2} \times t)$. This is clear that level $l_k$ decreases as $k$ itself decreases. Hence by monoticity in previous steps, we have $\lambda_{l_k} \geq \lambda_{l_{k+1}}$. As required, it follows that

$$\lambda_k \geq \min(\ \lambda_{l_{k+1}} \times (1 + \frac{1}{2} \times t)\ ,\ \frac{\epsilon}{2LS_{k+1}}\ ).$$

To finish the proof, we need to show inequalities on the right in $(*)$. This is a straightforward consequence of the choice of $\lambda_k$ in LLS. Indeed, $\lambda_k$ is calibrated to $\lambda_{l_k} \times (1 + \frac{1}{2} \times t)$ which is, by definition of $l_k$, the lowest possible value of $\lambda_l \times (1 + \frac{1}{2} \times t)$ among possible upper neighbors $l \sim k$, $l > k$. ■

## 8.3 A proof for theoretical estimation in the case of well-oredered databases

Our utility analysis of LLS mechanism relies on Proposition 12, which is the case of well-ordered databases.

**Proof of Proposition 12** Since we consider well-ordered databases for which sensitivities always satisfy $\frac{LS_{k+1}}{LS_k} \geq 1 + \frac{t}{2}$, each step in Algorithm 1 is a *jump*, that is we define $\lambda_k$ to be $\min(\frac{\epsilon}{2LS_k}, \lambda_{l_k} \times (1 + \frac{t}{2}))$ for each $1 \leq k \leq r$. Moreover, since instances in level $D_k$ can have neighbors only in $D_{k-1}$ or $D_{k+1}$, it is clear that we have $\tau_k = k - 1$ and $l_k = k + 1$ for all $k$. In particular, we have $\lambda_{l_k} = \lambda_{k+1}$ and then $\lambda_k = \min(\frac{\epsilon}{2LS_k}, \lambda_{k+1} \times (1 + \frac{t}{2}))$.

Now notice that the inequalities $\lambda_{k+1} \times (1 + \frac{t}{2}) \leq \frac{\epsilon}{2LS_k}$ for all $k$. Indeed, this follows from a simple induction and the following inequalities:

$$\begin{aligned}\lambda_{k+1} \times (1 + \frac{t}{2}) &\leq \lambda_{k+1} \times \frac{LS_{k+1}}{LS_k} \\ &\leq \frac{\epsilon}{2LS_{k+1}} \times \frac{LS_{k+1}}{LS_k} \\ &= \frac{\epsilon}{2LS_k}.\end{aligned}$$

Hence we have $\lambda_k = \lambda_{k+1} \times (1 + \frac{t}{2})$ for all $k$. The result follows from a straightforward induction. ■

## 8.4 Relationship with Smooth Sensitivity calibrated noise

In [16], the authors introduced the notion of Smooth Sensitivity, a notion of sensitivity between the local sensitivity and the global sensitivity, designed to achieve private algorithms with better accuracy than those calibrated on the global sensitivity. Smooth Sensitivity is a particular case of the following notion of $\beta$-smooth upper bound.

**Definition 16** (**$\beta$-smooth upper bound**). Let $\beta > 0$, and let $f : \mathcal{D} \to R$ be a query. A function $S : \mathcal{D} \to \mathbb{R}^+$ is a $\beta$-smooth upper bound on the local sensitivity $LS(f)$ if the following conditions hold for all $x \subset \mathcal{D}$:
(i) $S(x) \geq LS(f)(x)$;
(ii) $S(x) \leq e^\beta \times S(x')$ for all $x' \sim x$.

Global sensitivity $GS(f)$ is a (constant) 0-upper bound on $LS(f)$. The $\beta$-smooth sensitivity is another example of $\beta$-smooth upper bound.

**Definition 17** (**$\beta$-smooth sensitivity**). Let $\beta > 0$, and let $f : \mathcal{D} \to \mathbb{R}$ be a query. The $\beta$-smooth sensitivity $S^*_{f,\beta}$ of $f$ is defined by

$$S^*_{f,\beta}(x) = \max_{y \subset \mathcal{D}} \ (\ LS(f)(y) \times e^{-\beta d(x,y)}\ )$$

where $d(x, y)$ is the number of individuals on which the databases $x$ and $y$ differ.

The authors of [16] show that $\beta$-smooth sensitivity of a query $f$ is the optimal $\beta$-smooth upper bound on $LS(f)$. Lemma 2.5 in [16] states that the previous instance-based noise is $(\epsilon, \delta)$-differentially private for $\lambda_x = \frac{\epsilon}{2 \times S(x)}$, where $S$ is a $\beta$-smooth upper bound on local sensitivity with $\beta = \frac{\epsilon}{2 \times \ln(1/\delta)}$.

It appears that Lemma 2.5 in [16] is an easy consequence of our new Theorem 8, for a suitable choice of parameters $\alpha_x, \lambda_x$. Indeed, let $S$ be a $\beta$-smooth upper bound on $LS(f)$, and fix $\beta = \frac{\epsilon}{2\times\ln(1/\delta)}$. Take $\alpha_x = \frac{1}{2}$ and $\lambda_x = \frac{\epsilon}{2\times S(x)}$. For $\beta > 0$, we have $1 - e^{\beta} \leq \beta$. Since $S$ is a $\beta$-smooth upper bound on local sensitivity, we have for all $x' \sim x$,

$$\begin{aligned}|1 - \frac{S(x)}{S(x')}| &\leq |1 - e^{\beta}| \\ &\leq \beta \\ &\leq \frac{\epsilon}{2} \times \frac{1}{\ln(1/\delta)}.\end{aligned}$$

Hence condition 2 of the assumptions statement in Theorem 8 is satisfied. Moreover, condition 1 is straightforward since $S$ is an upper bound on local sensitivity $LS$. Then, Theorem 8 asserts that $(\epsilon, \delta)$-differential privacy is guaranteed by this choice of parameters.

## 8.5 More details about Section 5.2

First we give the proof of Proposition 15.

**Proof of Proposition 15** Recall that for a graph $H$, the upper bound $\overline{LS}(H) = \frac{5}{2}\times a_H$ on local sensitivity $LS(H)$ is obtained from $a_H = \max_{e\in H+e', e'\notin H} a_e^{H+e'}$, where $a_e^{H+e'}$ denotes the number of triangles in graph $H + e'$, involving edge $e$.

We need to show $|a_H - a_{H'}| \in \{0, 1\}$ for all $H \sim H'$. So let $H, H'$ be two neighbor graphs. Without loss of generality, we can assume that $H' = H + e$ for some edge $e \notin H$. It is straightforward that the inequality $a_H \leq a_{H'}$ holds.

To prove the other required inequality, note that proving the following assertion is sufficient: for all pair of edges $(e_1, e_2)$, $e_2 \notin H'$ there exist a pair $(e_3, e_4)$, $e_4 \notin H$ such that $a_{e_1}^{H'+e_2} \leq a_{e_3}^{H+e_4} + 1$.

Let $(e_1, e_2)$ be a pair of edges such that $e_2 \notin H'$. Set $e_3 = e_1$ and $e_4 = e_2$. Note that $e_4 \notin H$ since we have $e_2 \notin H'$ and $H \subset H'$. The result follows from a simple geometric fact: adding an edge to $H + e_2$ (to obtain $H' + e_2$) can increase the number of triangles involving $e_3$ by at most 1. Indeed, two situations can occur:

1. The graph $H + e_2$ contains at least one pair of adjacent edges of the form $(e_3, e'')$. In that case, the edge $e$ adds one triangle if it completes one f the pairs $(e_3, e'')$, or does not impact the number of triangles involving $e_3$ otherwise.

2. Edge $e_3$ is not involved in such a pair in graph $H + e_2$. In that case, the addition of edge $e$ has no impact on the number of triangles involving $e_3$.

■

In figure 9, we provide details on how mechanism LLS performs its computation on the basic example introduced in Section 5.2.

```
Algorithm LLS
**Input**: graph $G$, query $f_{2\Delta}$,
privacy parameters $\epsilon, \delta$
**Output**: private value for $f_{2\Delta}(G)$
. set $t = \frac{\epsilon}{\ln(1/\delta)}$
. $\overline{LS}(H) = 15/2$
. $LS_4 = 10$ and $\lambda_4 = \frac{\epsilon}{20}$
. $\overline{LS}(H) \neq LS_4$
. $LS_3 = 15/2$
. $\tau_4 = 3$
. $LS_2 = 5$
. $\tau_3 = 2$
. **if** $\frac{LS_4}{LS_3} < (1 + \frac{1}{2} \times t)$,
. $\lambda_3 = \frac{\epsilon}{20}$
. **else** $l_3 = 4$
. $\lambda_3 = \min(\frac{\epsilon}{15}, \frac{\epsilon}{20} \times (1 + \frac{t}{2}))$
. **end if**
. $\overline{LS}(H) = LS_3$, **return** $3 + \mathrm{Lap}(\lambda_3)$
```

Figure 9: Our instance-based noise algorithm LLS on a simple graph query