Local dampening: differential privacy for non-numeric queries via local sensitivity

Victor A. E. Farias 1 · Felipe T. Brito 1 · Cheryl Flynn 2 · Javam C. Machado 1 · Subhabrata Majumdar 3 · Divesh Srivastava 2

1 Introduction

Recent regulations on data privacy, such as General Data Protection Regulation (GDPR) [1] and Lei Geral de Proteção de Dados Pessoais (LGPD) [2], pose strict privacy requirements when gathering, storing and sharing data. Specifically, they require that an individual’s information be rendered
anonymous so that the individual is no longer identifiable from the published information.

Differential privacy [3,4] is the state-of-the-art formal definition for data release under strong privacy guarantees. It imposes near-indistinguishability on the released information whether an individual belongs to a sensitive database or not. The key intuition is that the output distribution of a differentially private query should not change significantly based on the presence or absence of an individual. It provides statistical guarantees against the inference of private information through the use of auxiliary information.

Algorithms can achieve differential privacy by employing output perturbation, which releases the true output of a given non-private query \( f \) with noise added. The magnitude of the noise should be large enough to cover the identity of the individuals in the input database \( x \).

For a numeric query (i.e., query with numeric output) \( f \), the Laplace mechanism [4] is a well-known output perturbing private method. It adds numeric noise to the output of \( f \) and calibrates the noise based on \( f \) and not on \( x \). The noise magnitude is proportional to the concept of global sensitivity, which measures the worst-case impact on \( f \)'s output of the addition or removal of an individual over the set of possible input databases. This may result in an unreasonably high amount of noise when \( x \) is far from the database with the worst-case impact, which is the case for many realistic databases. To remedy this, Nissim et al. [5] proposed to add instance-based noise calibrated as a function of \( x \). They introduced the notion of local sensitivity, which quantifies the impact of addition or removal of an individual for the database instance \( x \), resulting in a lower bound to the global sensitivity. Many works use this notion to shrink the amount of noise added to release more useful statistical information [6–10].

For the class of non-numeric queries \( f \), i.e., \( f \) has a non-numeric range \( \mathcal{R} \), the exponential mechanism [11] ensures differential privacy by sampling elements from \( \mathcal{R} \) using the exponential distribution. This requires a utility function \( u(x, r) \) that takes as input a database \( x \) and an element \( r \in \mathcal{R} \) and outputs a numeric score that measures the utility of \( r \). The larger \( u(x, r) \), the higher the probability of the exponential mechanism outputting \( r \). The exponential mechanism uses a similar notion of global sensitivity to that found in [4] where it measures the worst-case impact on the utility \( u(x, r) \) for all elements \( r \in \mathcal{R} \) by adding or removing an individual from all databases. However, to the best of our knowledge, the literature lacks generic mechanisms that apply local sensitivity to the non-numeric setting. Example 1 introduces the running example of this paper and discusses the computation of global sensitivity.

**Example 1** Consider an application where, given a graph \( G \), the query should report the node with the largest *Egocentric* Betweenness Centrality (EBC) [12–14]. The EBC metric measures the degree to which nodes stand between each other, defined as

\[
EBC(c) = \sum_{u,v \in N_c | u \neq v} \frac{g_{uv}(c)}{g_{uv}}.
\]

where \( g_{uv} \) is the number of geodesic paths connecting \( u \neq c \) and \( v \neq c \) on the subgraph composed by \( c \) and its neighbors \( N_c \) and \( g_{uv}(c) \) is the number of those paths that include \( c \).

For instance, let \( G \) be the graph illustrated in Fig. 1a. Node \( a \) has EBC score equal to 7.5 since there are \( \binom{9}{2} = 15 \) pairs of neighbors of the form \( \{v_i, v_j\} \), for \( 0 \leq i < j \leq 5 \), that each contributes with 0.5 as they have two geodesic paths of length 2 from \( v_i \) to \( v_j \), where only one contains \( a \). Pairs of the form \( \{b, v_i\} \), for \( 0 \leq i \leq 5 \) do not contribute to the score of \( a \) since there is only one geodesic path (length 1) from \( b \) to \( v_i \) and it does not contain \( a \). Verify that for the graph illustrated in Fig. 1b, node \( a \) has EBC score equal to 6.5.

We use edge differential privacy where the sensitivity is measured by adding or removing one edge from the input graph. The global sensitivity for EBC is obtained from a graph of the form displayed in Fig. 1a. Removing the edge \((a, b)\) is the worst case. The new EBC score of \( a \) is 15, 1 point for each one of the 15 pairs \( \{v_i, v_j\}, 0 \leq i < j \leq 5 \), as now there is only one geodesic path from \( v_i \) to \( v_j \) which includes \( a \) (path \( \{v_i, a, v_j\} \)). The paths of the form \( \{b, v_i, v_j\} \) are not counted since \( b \) no longer belongs to \( N_a \).

Note this gadget that sets the global sensitivity is formed by two nodes with high degree that share all neighbors and those neighbors do not have an edge to each other. Thus, this gadget is very unlikely to be found in real graphs. Figure 1b illustrates a more typical example. In this instance, the worst local measurement of the sensitivity is given by the removal of the edge \((a, v_0)\) that shrinks the EBC score of \( a \) by only 3 (1 for each pair \{v_0, b\}, \{v_0, v_2\} and \{v_0, v_3\} since \( v_0 \) is no longer a neighbor of \( a \)).

Additionally, the global sensitivity of EBC grows quadratically with respect to the maximum degree as discussed in Sect. 6. Hence a non-numeric mechanism applying local sensitivity could add less noise to the output compared to a global sensitivity-based approach.
In this paper, we propose the local dampening mechanism, which adapts the notion of local sensitivity to the non-numeric setting and uses it to dampen the utility function $u$ in order to increase the signal-to-noise ratio. Local dampening also employs the exponential distribution as the exponential mechanism [11]. Applications in which local sensitivity is significantly smaller than global sensitivity can benefit from our approach. For the scenario where local sensitivity is near the global sensitivity, the local dampening mechanism reverts to the exponential mechanism, so that the maximum addition of noise across all values of $r$ is bounded by the exponential mechanism.

To this end, we present a new version of the local sensitivity, called element local sensitivity. Traditional local sensitivity measures the largest impact of the addition or deletion of an individual to the input database over all outputs $r \in \mathcal{R}$. Element local sensitivity computes this impact, but only for some given element $r \in \mathcal{R}$. This allows us to explore local measurements of the sensitivity of $f$ even if traditional local sensitivity is near the global sensitivity, but, for most elements in $\mathcal{R}$, the element local sensitivity is low.

**Example 2** Consider the graph in Fig. 1a. The removal of the edge $(a, b)$ sets the traditional local sensitivity to 7.5 which is equal to the global sensitivity. But measurements of sensitivity per node (element) are much smaller. For example, the sensitivity for a node $v_i$ ($0 \leq i \leq 5$) is 1 which is set by the removal of edge $(a, b)$ where $EBC(v_i)$ increases from 0 to 1 (path $(a, v_i, b)$). We explore this to improve the accuracy further.

We illustrate the effectiveness of the local dampening mechanism by applying it to three diverse problems: (i) Percentile selection problem, where the task is to release the label of the $p$-th percentile element; (ii) Influential Node analysis, which searches for central nodes in a graph database. Given a centrality/influence metric, we release the label of the top-k most central nodes while preserving the privacy of the relationships between nodes in the graph; (iii) We also provide an application on tabular data that is a private adaptation to the ID3 algorithm to build a decision tree from a given tabular dataset based on the information gain for each attribute.

Our contributions are summarized as follows:

- We adapt the local sensitivity definition to the non-numeric setting (Sect. 2), and we introduce a new definition of local sensitivity that measures sensitivity per element (Sect. 3).
- We introduce the Local Dampening mechanism, a novel differentially private mechanism to answer non-numeric queries that applies local sensitivity to attenuate the utility function to increase the signal-to-sensitivity ratio to reduce noise (Sect. 3).
- We present a second version of our approach which we call the Shifted Local Dampening mechanism, which can effectively use the element local sensitivity to improve accuracy (Sect. 4).
- We develop a theoretical and empirical accuracy analysis where we enumerate some conditions under which the local dampening mechanism benefits from the local sensitivity notions. Under those conditions, we show that the exponential mechanism is an instance of the local dampening mechanism, and it is the worst instance of the local dampening mechanism in terms of accuracy. Also, we discuss the scenario where those conditions are not met and how we can still have good accuracy (Sect. 4.4).
- We tackle the Percentile Selection problem where a private mechanism should report the label of the $p$-th percentile element. Empirical results show that the local dampening mechanism can improve up to 73% over global sensitivity-based approaches (Sect. 5).
- We apply the local dampening mechanism to construct differentially private algorithms for a graph problem called Influential Node Analysis using Egocentric Betweenness Centrality as the influence metric, and we show how to compute local sensitivity for this application. Experimental results show that our approach could be as accurate as global sensitivity-based mechanisms using 2 to 4 orders of magnitude less privacy budget than global sensitivity-based approaches. Additionally, we perform an empirical scalability analysis of the proposed algorithm and show a sub-quadratic runtime behavior (Sect. 6).
- We address the application of building private algorithms for decision tree induction as an example data mining application for tabular data. We present a differentially private adaptation of the entropy-based ID3 algorithm using the local dampening mechanism, and we provide a way to compute the local sensitivity efficiently. We improve accuracy up to 12% compared to previous works (Sect. 7).

Section 2 presents the relevant formal definitions for Differential Privacy. Related work is presented in Sects. 3.6, 6.4 and 7.2. Finally, Sect. 8 concludes the paper. We defer the proofs of the lemmas and theorems to our technical report [15], except Theorem 3 where its proof is provided in this paper. We defer all the proofs to the appendix, except Theorem 3 where its proof is provided in this paper.

## 2 Differential privacy

Let $x$ be a sensitive database and $f$ a function to be evaluated on $x$. The database is represented as vector $x \in \mathcal{D}^n$ where each entry represents an individual tuple. The output $f(x)$
must be released without leaking much information about the individuals. For that, we need to design a randomized algorithm \( \mathcal{M}(x) \) that adds noise to \( f(x) \) such that it satisfies the definition of differential privacy stated below.

**Definition 1** (\( \epsilon \)-Differential Privacy [4,16]). A randomized algorithm \( \mathcal{M} \) satisfies \( \epsilon \)-differential privacy, if for any two databases \( x \) and \( y \) satisfying \( d(x, y) \leq 1 \) and for any possible output \( O \) of \( \mathcal{M} \), we have

\[
\Pr[\mathcal{M}(x) = O] \leq \exp(\epsilon) \Pr[\mathcal{M}(y) = O],
\]

where \( \Pr[\cdot] \) denotes the probability of an event and \( d \) denotes the hamming distance between the two databases, i.e., the number of tuples of individuals that changed value, i.e.,

\[
d(x, y) = | \{ i | x_i \neq y_i \} |.
\]

We refer to \( d \) as the distance between two given databases.

The parameter \( \epsilon \) dictates how close the distribution of the outputs differs between the databases \( x \) and \( y \). Small values of \( \epsilon \) mean that those two distributions must be really close which hurts accuracy but provides a better indistinguishability, i.e., a better privacy level. For large \( \epsilon \), the opposite happens, the two distributions can differ more which means a better accuracy and a lower level of privacy. The data querier may want to submit multiple queries to the database. The parameter \( \epsilon \) is also called as the privacy budget since each query submitted to the database consumes a part of the budget (refer to Sect. 2.3).

### 2.1 The non-numeric setting

We consider two settings when building a private mechanism: the numeric setting and the non-numeric setting. The numeric setting is when one needs to construct a private mechanism for a function \( f : \mathcal{D}^n \rightarrow \mathcal{R} \) where \( f \) outputs a vector of numeric values, i.e., \( \mathcal{R} = \mathbb{R}^d \). In this case, the Laplace Mechanism applies [4].

In this work, we address the non-numeric setting. We aim to build a private mechanism for a function \( f : \mathcal{D}^n \rightarrow \mathcal{R} \) where \( f \) outputs non-numeric values, i.e., \( \mathcal{R} \) denotes a discrete set of outputs \( \mathcal{R} = \{ r_1, r_2, r_3, \ldots \} \).

In the non-numeric setting, the data querier needs to provide a utility function \( u : \mathcal{D}^n \times \mathcal{R} \rightarrow \mathbb{R} \) that takes a database \( x \) and an output \( r \in \mathcal{R} \) and produces an utility score \( u(x, r) \). The utility function is application-specific and each application requires its own utility function.

The utility score represents how good an output \( r \) is for the dataset \( x \). This means that, for a given input database \( x \), the analyst prefers that the mechanism outputs the elements with high utility score. Thus a mechanism for answering \( f \) needs to output a high utility output with higher probability.

### 2.2 Sensitivity in the non-numeric setting

Differentially private mechanisms usually perturb the true output with noise. The amount of noise added to the true output of a non-numeric function \( f : \mathcal{D}^n \rightarrow \mathcal{R} \) is proportional to the sensitivity of the utility function \( u : \mathcal{D}^n \times \mathcal{R} \rightarrow \mathbb{R} \). The various notions of sensitivity used in this work are presented in this section.

**Global sensitivity.** The global sensitivity of \( u \) is defined as the maximum possible difference of utility scores at all possible pairs of database inputs \( x, y \) and all possible elements \( r \in \mathcal{R} \):

\[
\Delta u = \max_{r \in \mathcal{R}} \max_{x, y \mid d(x, y) \leq 1} | u(x, r) - u(y, r) |.
\]

The global sensitivity measures the maximum change in the utility score over all \( r \in \mathcal{R} \) between all pair of neighboring databases in the set of all databases.

**Local sensitivity** In non-numeric mechanisms, a larger global sensitivity \( \Delta u \) implies a lower accuracy. Thus, when constructing private solutions, we seek mechanisms with low sensitivity. The concept of local sensitivity \( LS(x) \) [5] captures the sensitivity locally on the input database \( x \) instead of searching for the sensitivity in the universe of databases \( \mathcal{D}^n \).

Local sensitivity tends to be smaller than global sensitivity for many problems [5–10]. In those problems, the real-world databases are very different from the worst-case scenario of the global sensitivity and they have a low observed local sensitivity. We present an adapted version of the definition of local sensitivity for the non-numeric setting, as given in [5].

**Definition 3** (Local Sensitivity, adapted from [5]) Given a utility function \( u(x, r) \) that takes as input a database \( x \) and an element \( r \) and outputs a numeric score, the local sensitivity of \( u \) is defined as

\[
LS^u(x) = \max_{r \in \mathcal{R}} \max_{y \mid d(x, y) \leq t} | u(x, r) - u(y, r) |
\]

Observe that the global sensitivity is the maximum local sensitivity over the set of all databases, \( \Delta u = \max_x LS^u(x) \).

However, using solely the local sensitivity to build a mechanism is not enough to satisfy differential privacy. Thus, like the smooth sensitivity framework [5], a part of our solution is based on local sensitivity at distance \( t \). We adapt the notion of local sensitivity at distance \( t \) [5] to the non-numeric setting for use in this work:
Local dampening: differential privacy for non-numeric queries via local sensitivity

2.3 Composition

The data querier can pose several queries to the database to compose complex differentially private algorithms. In this work, we focus on two types of composition: sequential composition and parallel composition. The sequential composition applies when a set of mechanisms is executed against a dataset. This implies that the privacy budget used on each computation sums up:

**Theorem 1** (Sequential composition [11,17]) Let $\mathcal{M}_i : \mathcal{D}^n \rightarrow \mathcal{R}_i$ be an $\epsilon_i$-differentially private algorithm for $i \in [k]$. Then $\mathcal{M}(x) = (\mathcal{M}_1(x), \ldots, \mathcal{M}_k(x))$ is $(\sum_{i=1}^{k} \epsilon_i)$-differentially private.

The parallel composition applies if queries are applied to disjoint subsets of the database. Suppose that the data querier issues many $\epsilon_i$-differentially private mechanisms over disjoint subsets of the database. It composes to a $\max_i \epsilon_i$-differentially private mechanism. This scenario has a lower privacy cost compared to the sequential composition.

**Theorem 2** (Parallel composition [17]) Let $\mathcal{M}_i : \mathcal{D}^n \rightarrow \mathcal{R}_i$ be an $\epsilon_i$-differentially private algorithm for $i \in [k]$ and $x_1, \ldots, x_k$ be pairwise disjoint subsets of $\mathcal{D}^n$. Then $\mathcal{M}(x) = (\mathcal{M}_1(x_1), \ldots, \mathcal{M}_k(x_k))$ is $(\max_i \epsilon_i)$-differentially private.

3 Local dampening mechanism

We present the local dampening mechanism, an output perturbing differentially private mechanism for the non-numeric setting that uses local sensitivity to reduce the noise injected to the true answer. Our approach uses the local sensitivity notions described in the Sect. 2.2. Additionally, we introduce a new notion of sensitivity called element local sensitivity. It measures the worst impact on the sensitivity for a given element $r \in \mathcal{R}$ when adding or removing an individual from the input database $x$, i.e., the largest difference $|u(x, r) - u(y, r)|$ for all neighbors $y$ of $x$.

More broadly, we coin the notion of sensitivity function that generalizes the all local sensitivity definitions. This concept is specially useful when computing the sensitivity is not possible or efficient but computing an upper bound is simpler, as it can be NP-hard [5,10].

The local dampening mechanism employs a sensitivity function to dampen the utility function $u$ and construct its dampened version, referred to $D_u(x, \Delta u)$. Specifically, we attenuate $u$ such that the signal-to-sensitivity ratio (i.e., $u$/sensitivity) is larger which results in higher accuracy. A sensitivity function is a function that computes one of the notions of sensitivity or an upper bound on the sensitivity.

3.1 Element local sensitivity

The local sensitivity at distance $t$, $LS^u(x, t)$, quantifies the maximum sensitivity of $u$ over all elements $r \in \mathcal{R}$ for an input database $x$ with $t$ modifications (Definition 4). That gives a high-level description of the variation of $u$ in neighboring databases. However, if just one element in $\mathcal{R}$ has a high value of sensitivity (close to $\Delta u$), $LS^u(x, t)$ will be equally large. That is ineffective in a scenario where most of the elements have low sensitivity and just few have high sensitivity, which makes $LS^u(x, t)$ large and consequently hurts accuracy.

We introduce a more specialized definition of local sensitivity named element local sensitivity, denoted as $LS^u(x, t, r)$, which measures the sensitivity of $u$ for a given
r ∈ ℜ for an input database x at distance t (Definition 5). This allows us to grasp the sensitivity of u for a single element.

**Definition 5 (Element Local Sensitivity at distance t).** Given a utility function u(x, r) that takes as input a database x and an element r and outputs a numeric score for x, the element local sensitivity at distance t of u is defined as

\[ LS_u(x, t, r) = \max_{y \in ℑ^d|d(x,y)\leq t, z \in ℑ^d|d(y,z)\leq 1} |u(y, r) - u(z, r)|, \]

where d(x, y) denotes the distance between two databases.

Note that we can obtain LS_u(x, t) from this definition: LS_u(x, t) = \max_{y \in ℑ^d} LS_u(x, y, t, r) as LS_u(x, t, r) = \max_{y \in ℑ^d|d(x,y)\leq t} LS_u(y, 0, r).

**Example 4 (Element local sensitivity)** We illustrate this definition with the same setup from previous examples. Let G be the graph from Fig. 2a. Suppose we want to compute the element local sensitivity for v4, LS_u(G, 0, v4). We measure only the worst impact of the addition or removal of an edge on the value of the EBC score for v4. This is obtained by adding the edge (v0, v4) (Fig. 3). The EBC score increases by 2 (1 for each pair [b, v0] and [v0, v5]). Thus LS_u(G, 0, v4) = 2 which is smaller than local sensitivity LS_u(G, 0) = 3 and ∆u = 7.5 as discussed in Example 1.

### 3.2 Sensitivity functions

Computing local sensitivity LS_u(x, t) or element local sensitivity LS_u(x, t, r) is not always feasible, as it can be NP-hard [5,10]. To navigate this problem, we can relax the need for the computation of LS_u(x, t) or LS_u(x, t, r) and build a computationally efficient function δ_u(x, t, r) that computes an upper bound for LS_u(x, t) or LS_u(x, t, r) that is still smaller than ∆u. We refer to δ_u as a sensitivity function that has the following signature δ_u : ℑ^d × ℑ^d × ℜ → ℜ. Note that δ_u(x, t, r) = LS_u(x, t, r) or δ_u(x, t, r) = LS_u(x, t) are sensitivity functions.

We define a classification of sensitivity functions based on four properties: admissibility, boundedness, monotonicity and stability.

#### 3.2.1 Admissibility

The sensitivity function δ_u needs to have some properties to be admissible in the local dampening mechanism to guarantee differential privacy:

**Definition 6 (Admissibility)** A sensitivity function δ_u(x, t, r) is admissible if:

1. δ_u(x, 0, r) ≥ LS_u(x, 0, r), for all x ∈ ℑ^d and all r ∈ ℜ
2. δ_u(x, t + 1, r) ≥ δ_u(y, t, r), for all x, y such that d(x, y) ≤ 1 and all t ≥ 0

The global sensitivity ∆u is admissible as ∆u ≥ LS_u(x, 0, r), for all x and a constant value would satisfy the second requirement of Definition 6. We also show that the function LS_u(x, t, r) itself is admissible.

**Lemma 1** The element local sensitivity LS_u(x, t, r) is admissible.

A intermediate result shows (Lemma 2) that LS_u(x, t) is an admissible function once LS_u(x, t) = max_{x \in ℑ^d} LS_u(x, t, r) and LS_u(x, t, r) is an admissible function (Theorem 1).

**Lemma 2** Let δ_1(x, t, r),..., δ_p(x, t, r) be admissible functions. Then δ(x, t, r) defined as δ(x, t, r) = max{δ_1(x, t, r),..., δ_p(x, t, r)} is an admissible sensitivity function.

The proof of Lemma 2 is immediate given by the admissibility of δ_1(x, t, r),..., δ_p(x, t, r).

#### 3.2.2 Boundedness

Some sensitivity functions, such as LS_u(x, t) and LS_u(x, t, r), converge to ∆u, by design, as t grows. This follows from the fact that the maximum distance of two databases is at most n by the hamming distance definition. Thus when t = n, LS_u(x, t) and LS_u(x, t, r) measure sensitivity in all possible databases. We refer to those functions as bounded functions.

Note that one can force a given function δ_u(x, t, r) to be bounded by simply replacing it by its bounded version min(δ_u(x, t, r), ∆u). We show that min(δ_u(x, t, r), ∆u) is admissible and bounded.

**Lemma 3** If δ_u(x, t, r) is admissible, then min(δ_u(x, t, r), ∆u) is admissible and bounded.

**Definition 7 (Boundedness)** A sensitivity function δ_u(x, t, r) is said to be bounded if δ_u(x, t, r) = ∆u for all t ≥ n.

Thus, we can replace δ_u(x, t, r) with min(δ_u(x, t, r), ∆u) since its admissible. In terms of accuracy, this replacement is beneficial. We have that δ_u(x, t, r) ≥ min(δ_u(x, t, r), ∆u) for all database x, t ≥ 0 and r ∈ ℜ. Thus δ_u(x, t) is

![Diagram](https://via.placeholder.com/150)
always larger than \( \min(\delta^n(x, t, r), \Delta u) \) meaning that local dampening injects less noise as sensitivity is proportional to the noise. This means that we can impose boundedness for any function and, beyond that, we have gains in accuracy as it injects less noise.

### 3.2.3 Monotonicity

We introduce the notion of monotonicity in our context. When the utility score \( u(x, r) \) is a monotonic function of \( \delta^n(x, t, r) \) over \( r \in \mathcal{R} \), we say that \( \delta^n(x, t, r) \) is monotonic. We have three classifications for monotonicity.

**Definition 8 (Non-decreasing Monotonicity)** Let \( u(x, r) \) be a utility function and \( \delta^n(x, t, r) \) be a sensitivity function. \( \delta^n(x, t, r) \) is said to be monotonically non-decreasing if \( \delta^n(x, t, r') \geq \delta^n(x, t, r) \) for all \( x \in \mathcal{D}^n, r, r' \in \mathcal{R}, t \geq 0 \) such that \( u(x, r) \geq u(x, r') \).

And its symmetric definition is:

**Definition 9 (Non-increasing Monotonicity)** Let \( u(x, r) \) be a utility function and \( \delta^n(x, t, r) \) be a sensitivity function. \( \delta^n(x, t, r) \) is said to be monotonically non-decreasing if \( \delta^n(x, t, r) \geq \delta^n(x, t, r') \) for all \( x \in \mathcal{D}^n, r, r' \in \mathcal{R}, t \geq 0 \) such that \( u(x, r) \leq u(x, r') \).

Also, a sensitivity can be flat:

**Definition 10 (Flat Monotonicity)** Let \( u(x, r) \) be a utility function and \( \delta^n(x, t, r) \) be a sensitivity function. \( \delta^n(x, t, r) \) is said to be flat if \( \delta^n(x, t, r) = \delta^n(x, t, r') \) for all \( x \in \mathcal{D}^n, r, r' \in \mathcal{R}, t \geq 0 \).

We refer to a monotonic function as a function that is either flat, monotonically non-decreasing or monotonically non-increasing.

Note that flat sensitivity functions are independent on \( r \) and they are both monotonic non-increasing and monotonic non-decreasing. The global sensitivity \( \Delta u \) and the local sensitivity \( LS^n(x, t) \) are flat sensitivity functions since they do not depend on \( r \).

Additionally, given a utility function \( u \) and an admissible function \( \delta^n(x, t, r) \), one can build a function \( \hat{\delta}^n(x, t, r) \) from \( \delta^n(x, t, r) \) such that \( \hat{\delta}^n(x, t, r) \) is flat.

\[
\hat{\delta}^n(x, t, r) = \max_{r' \in \mathcal{R}} \delta^n(x, t, r').
\]

A drawback of using \( \hat{\delta}^n(x, t, r) \) is that \( \hat{\delta}^n(x, t, r) \geq \delta^n(x, t, r) \), for all \( x, t \geq 0 \) and \( r \in \mathcal{R} \) meaning that \( \hat{\delta}^n(x, t, r) \) returns a large upper bound for sensitivity and, consequently, hurts accuracy. Lemma 2 ensures that \( \hat{\delta}^n(x, t, r) \) is admissible if \( \delta^n \) is admissible.

### 3.2.4 Stability

**Definition 11 (Stability)** A sensitivity function \( \delta^n(x, t, r) \) is stable if \( \delta^n \) is admissible, bounded and monotonic.

Satisfying all three requirements (admissibility, boundedness and monotonicity) for designing a stable function may sound very restrictive. However, for all definitions of sensitivity, two of them are naturally stable: global sensitivity \( \Delta u \) and local sensitivity \( LS^n(x, t) \). Only the element local sensitivity \( LS^n(x, t, r) \) can be non-monotonic and, consequently, non-stable. Nevertheless, in Sect. 4.5, we argue that the requirement of strict monotonicity can be relaxed and an admissible bounded function with “weak” monotonicity can perform well in our approach.

Besides, for any sensitivity function, the requirement of boundedness can be easily imposed as shown in Sect. 3.2.2 while still providing lower sensitivity.

### 3.3 Damping function

A crucial part of our mechanism is the damping function. We now define the damping function \( D_{u, \delta^n}(x, r) \), which uses an admissible sensitivity function \( \delta^n(x, t, r) \) to return a dampened and scaled version of the original utility function.

**Definition 12 (Damping Function)** Given a utility function \( u(x, r) \) and an admissible function \( \delta^n(x, t, r) \), the damping function \( D_{u, \delta^n}(x, r) \) is defined as a piecewise linear interpolation over the points:

\[
(\ldots, (b(x, -1, r), -1), (b(x, 0, r), 0), (b(x, 1, r), 1), \ldots)
\]

where \( b(x, i, r) \) is given by:

\[
b(x, i, r) := \begin{cases} 
\sum_{j=0}^{i-1} \delta^n(x, j, r) & \text{if } i > 0 \\
0 & \text{if } i = 0 \\
-b(x, -i, r) & \text{otherwise}
\end{cases}
\]

Therefore,

\[
D_{u, \delta^n}(x, r) = \frac{u(x, r) - b(x, i, r)}{b(x, i + 1, r) - b(x, i, r)} + i
\]

where \( i \) is defined as the smallest integer such that \( u(x, r) \in [b(x, i, r), b(x, i + 1, r)] \).

Figure 4 shows the general scheme of \( D_{u, \delta^n} \). A crucial property of \( D_{u, \delta^n} \) is that it scales \( u \) so that the sensitivity of \( D_{u, \delta^n} \) is bounded to 1 (Lemma 4).

**Lemma 4** \( |D_{u, \delta^n}(x, r) - D_{u, \delta^n}(y, r)| \leq 1 \) for all \( x, y \) such that \( d(x, y) \leq 1 \) and all \( r \in \mathcal{R} \) if \( \delta^n \) is admissible.
We now state the local dampening mechanism a generic non-numerically differentially private mechanism. It takes a database \( x \), the privacy budget \( \epsilon \), the utility function \( u \), an admissible sensitivity function \( \delta^u \) and the range \( R \) of the function to be sanitized.

It samples an element from \( r \in R \) based on its dampened utility score \( D_u,\delta^u(x, r) \). The larger the score, the higher the probability of sampling it.

**Definition 13 (Local dampening mechanism)** The local dampening mechanism \( M_{LD}(x, \epsilon, u, \delta^u, R) \) selects and outputs an element \( r \in R \) with probability proportional to \( \exp\left(\frac{\epsilon D_u,\delta^u(x, r)}{2}\right) \).

This version of the local dampening mechanism is specially effective when the sensitivity function is flat. In the following example, we demonstrate the process operation of the local dampening mechanism.

**Example 5 (Local dampening mechanism)** This example explores the local dampening mechanism using the local sensitivity definition while the element local sensitivity is addressed in Sect. 4. Let \( G \) be the graph of Fig. 2a. As we have discussed in Example 2, we have that \( L^{EBC}(G) = 3 \) and \( L^{EBC}(G, 1) = 5 \). The EBC scores for the vertices are \( EBC(a) = EBC(b) = 6.5 \) and \( EBC(v_i) = 0 \), for \( 0 \leq i \leq 5 \). Their dampened EBC scores are:

\[
D_{EBC,L^{EBC}}(G, a) = D_{EBC,L^{EBC}}(G, b) = 1.7
\]

\[
D_{EBC,L^{EBC}}(G, v_i) = 0, \text{ for } 0 \leq i \leq 5
\]

For instance, assuming \( \epsilon = 2.0 \), the probability for each node to be selected is:

\[
\Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] \propto \exp(1.7) = 5.47
\]

\[
\Pr[v_i \text{ is selected}] \propto \exp(0) = 1.0, \text{ for } 0 \leq i \leq 5
\]

Normalizing, we have that \( \Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] = 0.32 \) and \( \Pr[v_i \text{ is selected}] = 0.06 \). Thus the local dampening mechanism samples a element with those probabilities.

### 3.5 Privacy guarantee

We now prove that the local dampening mechanism \( M_{LD} \) ensures \( \epsilon \)-differential privacy (Theorem 3). The privacy correctness proof follows from the exponential mechanism correctness [11] and Lemma 4.

**Theorem 3** \( M_{LD} \) satisfies \( \epsilon \)-Differential Privacy if \( \delta \) is admissible.

**Proof** Given two neighboring databases \( x, y \in D^n \) (i.e., \( d(x, y) \leq 1 \)) and an output \( r \in R \). We show that the ratio of the probability of \( r \) being produced by local dampening mechanism on database \( x \) and \( y \) is bounded by \( \exp(\epsilon) \).

\[
\frac{P_x(r)}{P_y(r)} = \frac{P[M_{LD}(x, u, R) = r]}{P[M_{LD}(y, u, R) = r]}
\]

\[
= \left( \frac{\exp(\frac{\epsilon D_{u,\delta^u}(x, r)}{2})}{\sum_{r' \in R} \exp(\frac{\epsilon D_{u,\delta^u}(x, r')}{2})} \right)
\]

\[
\times \left( \frac{\exp(\frac{\epsilon D_{u,\delta^u}(y, r)}{2})}{\sum_{r' \in R} \exp(\frac{\epsilon D_{u,\delta^u}(y, r')}{2})} \right)
\]

\[
\leq \exp\left(\frac{\epsilon}{2}\right) \cdot \exp\left(\frac{\epsilon}{2}\right)
\]

\[
\leq \exp(\epsilon)
\]

The two inequalities follow from lemma 4. By symmetry, \( \frac{P[M_{LD}(x, u, R) = r]}{P[M_{LD}(y, u, R) = r]} \geq \exp(-\epsilon) \) holds.

### 3.6 Related work

There is a vast literature on differential privacy for numeric queries, and we refer the interested reader to [18] for a recent survey. Our work was previously published in [19]. This paper advances on the understanding of the local dampening mechanism and on the related work. We discuss and carry out experiment with more recent related work, the permute-and-flip mechanism. We provide a new application to the local
dampening mechanisms and compare to the related work. We give a new theoretical accuracy analysis where we show how to compare two instances of the local dampening mechanism. Besides that, we show that, under some conditions, our approach is never worse than the exponential mechanism.

Given that, in this section, we discuss also the two available differential privacy approaches for the non-numeric, also known as the selection problem, setting in the literature, the exponential mechanism and the permute-and-flip mechanism.

3.6.1 Exponential mechanism

The exponential mechanism $M_{EM}$ [11] is the most used approach for providing differential privacy to the non-numeric setting. It uses a notion of global sensitivity $\Delta u$ (Definition 2).

The exponential mechanism privately answers a function $f : D^n \rightarrow R$ applied to database $x$ by sampling an element $r \in R$ with probability proportional to its utility score $u(x, r)$. It uses the exponential distribution to assign probabilities for each $r \in R$. The exponential mechanism is stated as follows:

**Definition 14** (Exponential Mechanism [11]). The exponential mechanism $M_{EM}(x, \epsilon, u, R)$ selects and outputs an element $r \in R$ with probability proportional to $\exp\left(\frac{\epsilon \cdot u(x,r)}{\Delta u}\right)$.

The exponential mechanism satisfies $\epsilon$-differential privacy [11]. In Sect. 4, we show that, under some conditions, the exponential mechanism is never worse than the local dampening in terms of accuracy. Additionally, we carry out an experimental evaluation with the two applications that we tackle in this work: influential node analysis (Sect. 6) and decision tree induction (Sect. 7).

**Example 6** (Comparison local dampening mechanism with exponential mechanism) We make a comparison of the probabilities of the local dampening mechanism with the exponential mechanism.

From Example 5, we have that $\Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] = 0.32$ and $\Pr[v_i \text{ is selected}] = 0.06$. While, according to Definition 14, the exponential mechanism obtained that $\Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] = 0.22$ and $\Pr[v_i \text{ is selected}] = 0.09$. Thus local dampening yields a higher probability of choosing the node with highest score.

3.6.2 Permute-and-flip

The *permute-and-flip* mechanism $M_{PF}$, [20] is recent work that also addresses differential privacy for the non-numeric setting. It is defined as an iterative algorithm that employs the exponential distribution to assign probabilities for each element $r$. Algorithm 1 defines permute-and-flip approach.

Basicly, the algorithm iterates over a random permutation of the elements $r \in R$ and then flip a biased coin with probability $\frac{u(x,r) - u^*}{}$. $u^*$ is the maximum utility observed over all elements in the range set $R$ given the input database $x$. Thus, the closer $u(x,r) - u^*$, more likely is $r$ to be outputted. The mechanism is guaranteed to terminate with a result because if $u(x,r) = u^*$, then the probability of heads is 1.

McKenna et al. [20] show that their approach is also never worse than the exponential mechanism in terms of accuracy. We conduct an empirical comparison of permute-and-flip mechanism to local dampening in Sects. 6 and 7.

4 Shifted local dampening mechanism

In this section, we present a second version of the local dampening mechanism name shifted local dampening mechanism $M_{SLD}$. This version is designed for non-flat monotonic sensitivity functions which is the most usual case in our experiments.

We develop an insightful discussion on accuracy of the shifted local dampening mechanism (Sect. 4.4). We provide tools to compare two instances of the shifted local mechanism in terms of accuracy. Also, these tools guide on the design of good sensitivity functions that provide accurate $M_{SLD}$ instances (Sect. 4.4.1). We show that, with a stable sensitivity function, the local dampening mechanism is never worse than the exponential mechanism (Sect. 4.4.2). Additionally, even if the stability condition is not met, we discuss how to construct good sensitivity functions (Sect. 4.5).

4.1 Inversion problem

First, we exemplify an issue that happens when the sensitivity function is not monotonic.

Consider the scenario where we dampen the utility scores of the elements $r \in R$ with the sensitivity function $\delta^n$ that is not monotonic. This might be the case when we...
use $\delta^u(x,t,r)$ as the element local sensitivity, $\delta^u(x,t,r) = LS^u(x,t,r)$.

In this situation, Example 7 illustrates a case where the local dampening changes the relative order of the dampened utility scores compared to the original utility scores. We refer to this problem as the inversion problem.

Example 7 (Inversion problem) Consider the following setup: $R = \{r_1, r_2\}$. $\delta^u(x,0, r_1) = 1$, $\delta^u(x,1, r_1) = 2$, $\delta^u(x,0, r_2) = 4$, $u(x,r_1) = 3$ and $u(x,r_2) = 4$. When applying $D_{u,\delta^u}$ to $r_1$ and $r_2$, we obtain $D_{u,\delta^u}(x,r_1) = 2$ and $D_{u,\delta^u}(x,r_2) = 1$. Originally, $r_2$ is more useful than $r_1$ but after dampening it inverts. This hurts accuracy since the local dampening mechanism will choose $r_1$ with higher probability.

4.2 Shifted local dampening

The key idea for this extension is the use of shifting in the utility score to take advantage of non-flat monotonic sensitivity functions $\delta^u$. The discussion in this section is focused on non-flat monotonic sensitivity functions. However, we show later that the shifted local dampening also performs well for non-strictly monotonic functions.

Example 8 (Utility function shifting) Consider the graph $G$ from Fig. 2a. For nodes $a$ and $b$, their measured element local sensitivities are: $LS^{EBC}(G,0,a) = LS^{EBC}(G,0,b) = 3$ and $LS^{EBC}(G,1,a) = LS^{EBC}(G,1,b) = 5$. For a node $v_i$, for $0 \leq i \leq 5$, its measured sensitivity is $LS^{EBC}(G,0,v_i) = 2$. We observe the non-decreasing monotonicity of $LS^{EBC}$, since the EBC scores are $EBC(a) = EBC(b) = 6.5$ and $EBC(v_i) = 0$, for $0 \leq i \leq 5$.

For instance, shifting the EBC scores by $-7$, we get that $EBC'(a) = EBC'(b) = -0.5$ and $EBC'(v_i) = -7$, for $0 \leq i \leq 5$. Then we compute their dampened EBC' scores:

$$D_{EBC',LS^{EBC}}(G,a) = D_{EBC',LS^{EBC}}(G,b) = 0.1$$

$$D_{EBC',LS^{EBC}}(G,v_i) = -2, \text{ for } 0 \leq i \leq 5$$

Let $\epsilon = 2.0$. The probability for each node to be selected is:

$$\Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] \propto \exp(0.1) = 0.44$$

$$\Pr[v_i \text{ is selected}] \propto \exp(-2) = 0.13, \text{ for } 0 \leq i \leq 5$$

Normalizing, we have that $\Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] = 0.472$ and $\Pr[v_i \text{ is selected}] = 0.0046$. Recall that, the exponential mechanism obtained that $\Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] = 0.22$ and $\Pr[v_i \text{ is selected}] = 0.09$ (Example 6) and, for the unshifted local dampening mechanism, (Example 5), we have that $\Pr[a \text{ is selected}] = \Pr[b \text{ is selected}] = 0.32$ and $\Pr[v_i \text{ is selected}] = 0.06$. The nodes with highest score increase probability compared to the unshifted local dampening and the exponential mechanism.

For the sake of argument, suppose that $\delta^u(x,t,r)$ is monotonically non-decreasing. We design the shifting in a way that it rearranges the utilities scores such that the distribution of the utility scores is more spread.

The idea is the following: we shift left enough so that all utility scores are negative. The elements with larger utility score are the elements with smallest absolute value after shifting. Thus, these shifted scores are dampened with large $\delta^u(x,t,r)$ (by assumption of non-decreasing monotonicity). This implies that large utility scores are dampened closer to 0 and the opposite happens to elements with small utility scores. The elements with small utility scores are dampened with small $\delta^u(x,t,r)$ and, consequently, the scores are less attenuated and far from 0. This results in more spread distribution of utility scores.

Hereby we propose to replace the original utility function $u$ with its shifted version $u^\epsilon$ where $s$ is the utility score shift and

$$u^\epsilon(x,r) = u(x,r) - s.$$

One could design a private query, consuming part of the privacy budget, to choose $s$ such that it minimizes some loss function to optimize accuracy. In this work, we set $s$ to a value that does not depend on private data, $s \to \infty$. In what follows, the shifted local dampening mechanism is stated as follows:

Definition 15 (Shifted Local Dampening Mechanism—non-decreasing sensitivity function). The shifted local dampening mechanism $M_{SLD}(x,\epsilon,u,\delta^u,\mathcal{R})$ outputs an element $r \in \mathcal{R}$ with probability equals to

$$\lim_{s \to \infty} \left( \frac{\exp \left( \frac{\epsilon D_{u,\delta^u}(x,r)}{2} \right)}{\sum_{r' \in \mathcal{R}} \exp \left( \frac{\epsilon D_{u,\delta^u}(x,r')}{2} \right)} \right).$$

When $\delta^u$ is monotonically non-increasing, the following definition of the shifted local dampening mechanism applies:

Definition 16 (Shifted Local Dampening Mechanism—non-increasing sensitivity function). The shifted local dampening mechanism $M_{SLD}(x,\epsilon,u,\delta^u,\mathcal{R})$ outputs an element $r \in \mathcal{R}$ with probability equals to

$$\lim_{s \to -\infty} \left( \frac{\exp \left( \frac{\epsilon D_{u,\delta^u}(x,r)}{2} \right)}{\sum_{r' \in \mathcal{R}} \exp \left( \frac{\epsilon D_{u,\delta^u}(x,r')}{2} \right)} \right).$$
For the case of functions that do not depend on \( r \), both versions of the shifted local dampening mechanism are applicable.

### 4.3 Privacy guarantee and implementation

We now prove that the shifted local dampening mechanism \( M_{\text{SLD}} \) ensures \( \epsilon \)-differential privacy. For the privacy guarantee, the sensitivity function \( \delta^u \) just needs to be admissible and bounded but not necessarily monotonic. Recall that boundedness can be easily achieved (Sect. 3.2.2).

We first show an intermediate result:

**Lemma 5** If \( \delta^u \) is an admissible and bounded sensitivity function then

\[
\sum_{r' \in \mathcal{R}} \frac{\exp(\epsilon \Delta u, \mu(x, r)/2)}{\sum_{r'' \in \mathcal{R}} \exp(\epsilon \Delta u, \mu(x, r'')/2)} \geq s \geq s_0 \text{ where } s_0 = n \Delta u + \max_{r' \in \mathcal{R}} u(x, r') \text{ and } n \text{ is the size of the input database.}
\]

Lemma also 5 gives hint about the implementation. It suffices to shift by \( n \Delta u + \max_{r' \in \mathcal{R}} u(x, r') \) to meet the definition of the shifted local dampening. Also, from Lemma 5, it follows directly (Corollary 1).

**Corollary 1** \( \lim_{s \to \infty} \left( \frac{\exp(\epsilon \Delta u, \mu(x, r)/2)}{\sum_{r' \in \mathcal{R}} \exp(\epsilon \Delta u, \mu(x, r')/2)} \right) \) exists and is equal to \( \sum_{r' \in \mathcal{R}} \frac{\exp(\epsilon \Delta u, \mu(x, r')/2)}{\sum_{r'' \in \mathcal{R}} \exp(\epsilon \Delta u, \mu(x, r'')/2)} \) for \( s \geq s_0 \) where \( s_0 = n \Delta u + \max_{r' \in \mathcal{R}} u(x, r') \) and \( n \) is the size of the input database.

The privacy correctness proof follows from the exponential mechanism correctness [11], Lemma 4 and Corollary 1. In this proof, we use the non-decreasing admissible function version of the local dampening (Definition 15). The non-increasing version (Definition 16) privacy guarantee proof is symmetric.

**Theorem 4** \( M_{\text{SLD}} \) satisfies \( \epsilon \)-Differential Privacy if \( \delta^u \) is admissible and bounded.

### 4.4 Accuracy analysis

In this section, we provide theoretical analysis on the accuracy. We aim to answer to the following questions: (i) How to compare two instances of the local dampening with two different admissible functions? (ii) Under which conditions does the local dampening performs more accurately than the exponential mechanism? (iii) If those conditions are not met, how to build good admissible functions? and (iv) How does local dampening compare to the exponential mechanism in terms of accuracy?

We evaluate the accuracy of a given mechanism \( \mathcal{M} \) by studying the error random variable \( \mathcal{E} \). \( \mathcal{E} \) gives how much the element sampled by \( \mathcal{M} \) differ from the optimal element in terms of utility.

\[
\mathcal{E}(\mathcal{M}, x) = u^* - u(x, \mathcal{M}(x))
\]

where \( u^* \) is the optimal utility score, \( u^* = \max_{r \in \mathcal{R}} u(x, r) \).

To compare two instances of the local dampening for the same problem, we need to analyze the features of the function \( \delta^u \). We develop a discussion on accuracy guarantees for stable functions where we show how to compare two stable functions and show that, using a stable function, the local dampening mechanism is never worse than the exponential mechanism in terms of accuracy.

#### 4.4.1 Accuracy analysis for stable sensitivity functions

Two instances of the local dampening mechanism can be compared by their stable sensitivity functions. As lower sensitivity means higher accuracy, a stable sensitivity function compared by their stable sensitivity functions. As lower sensitivity means higher accuracy, a stable sensitivity function.

**Definition 17** (Dominance) Let \( \delta^u(x, t, r) \) and \( \tilde{\delta}^u(x, t, r) \) be two stable sensitivity functions and \( x \) be a database. Let \( \alpha(x, t, r) \) refer to the gap between \( \delta^u(x, t, r) \) and \( \tilde{\delta}^u(x, t, r) \): \( \alpha(x, t, r) = \delta^u(x, t, r) - \tilde{\delta}^u(x, t, r) \). Assume that \( R = \{r_1, \ldots, r_q\} \) is ordered such that \( u(x, r_1) \geq \cdots \geq u(x, r_q) \). If \( \alpha(x, t, r_1) \geq \alpha(x, t, r_2) \geq \cdots \geq \alpha(x, t, r_q) \geq 0 \) for all \( t \geq 0 \), then \( \delta^u(x, t, r) \) dominates \( \tilde{\delta}^u(x, t, r) \).

Given that, we can affirm that an instance of the local dampening mechanism using \( \delta^u(x, t, r) \) is never worse than an instance using the dominated \( \tilde{\delta}^u(x, t, r) \):

**Lemma 6** (Shifted Local Dampening Accuracy) Let \( \tilde{\delta}^u(x, t, r) \) and \( \tilde{\tilde{\delta}}^u(x, t, r) \) be two stable functions and \( x \) be a database. If \( \tilde{\delta}^u(x, t, r) \) dominates \( \tilde{\delta}^u(x, t, r) \) then:

1. \( \Pr[\mathcal{E}(M_{\text{SLD}}, x) \geq \theta] \leq \Pr[\mathcal{E}(\overline{M}_{\text{SLD}}, x) \geq \theta] \) for all \( \theta \geq 0 \).
2. \( \mathbb{E}[\mathcal{E}(M_{\text{SLD}}, x)] \leq \mathbb{E}[\mathcal{E}(\overline{M}_{\text{SLD}}, x)] \).

where \( M_{\text{SLD}} \) represents an instance of the shifted local dampening mechanism using \( \delta^u \) as the sensitivity function while \( \overline{M}_{\text{SLD}} \) is an instance using \( \tilde{\delta}^u \).

We can use Lemma 6 as a tool understand the accuracy of the local dampening mechanism. It suggests that a sensitivity function should be as inclined as possible, i.e., a higher difference between two gaps \( \alpha(x, t, r_1) \) and \( \alpha(x, t, r_{i+1}) \) implies higher accuracy. Also, the gaps \( \alpha(x, t, r_1) \) should be as large as possible. To illustrate this, we provide an example:
Example 9 (Shifted Local Dampening Accuracy) Let \( \delta^u(\alpha, r, t) = (u(x, r), t) / \beta \). \( \Delta u \) be a sensitivity function where \( \beta \) is a parameter to control magnitude of \( \delta^u \) and, consequently, to control the accuracy of the shifted local dampening mechanism based on Definition 17 and Lemma 6.

Note that \( \delta^u_\beta \) is monotonically non-decreasing and can be bounded by the process depicted in Sect. 3.2.2. Also, assume that \( \delta^u \) is admissible for \( u \), thus it is a stable sensitivity function.

Without loss of generality, suppose that \( R = \{ r_1, \ldots, r_q \} \) is ordered such that \( u(x, r_1) \geq \cdots \geq u(x, r_q) \). The gaps \( \alpha(x, t, r_i) \) (Definition 17) between two sensitivity functions \( \delta^u_\beta \) and \( \delta^u \) are given by \( \alpha(x, t, r_i) = \delta^u_\beta(x, t, r_i) - \delta^u(x, t, r_i) = (u(x, r_i), t) / (1/\beta_1 - 1/\beta_2) \), for \( 1 \leq i \leq q \) and \( t \geq 0 \). In what follows, when \( \beta_1 \geq \beta_2 \), \( \delta^u_\beta \) dominates \( \delta^u \) as \( \alpha(x, t, r_i) = (u(x, r_i), t) / (1/\beta_1 - 1/\beta_2) \geq u(x, r_i+1), t) / (1/\beta_1 - 1/\beta_2) \geq \alpha(x, t, r_i+1) \) because \( u(x, r_i) \geq u(x, r_i+1) \) and \( (1/\beta_1 - 1/\beta_2) < 0 \) for \( t \geq 0 \).

This way, the parameter \( \beta \) also control the gaps \( \alpha(x, t, r_i) \) since it is proportional to the term \( (1/\beta_1 - 1/\beta_2) \). This way, we illustrate this by varying the parameter \( \beta \) while fixing the local sensitivity function \( \delta^u \).

4.4.2 Comparison to the exponential mechanism

A very useful property of both versions of the local dampening mechanism is that the exponential mechanism is an instance of the local dampening mechanism. The exponential mechanism is obtained by setting \( \delta^u(x, t, r) = \Delta u \) in an instance of the shifted local dampening (Lemma 7).

Lemma 7

\[
Pr[M_{EM}(x, \epsilon, u, R) = r] = Pr[M_{LD}(x, \epsilon, u, \delta^u, R) = r] = Pr[M_{SLD}(x, \epsilon, u, \delta^u, R) = r]
\]

Note that this result implies that we can use local sensitivity \( LS^u(x, t) \) safely since \( LS^u(x, t) \) is a stable sensitivity function, i.e., using the shifted local dampening mechanism with \( LS^u(x, t) \) as the sensitivity function is never worse than the exponential mechanism. Yet, it suggests that the larger the difference between \( LS^u(x, t) \) and \( \Delta u \), the more accurate it is in relation to the exponential mechanism.

This result also suggests that using \( \Delta u \) as a sensitivity function is the worst-case stable function. Given that, what would be the best stable function? The element local sensitivity \( LS^u(x, t, r) \) function is a good candidate. As shown before, \( LS^u(x, t, r) \) is admissible and bounded. However, \( LS^u(x, t, r) \) is not necessarily monotonic. We demonstrate that \( LS^u(x, t, r) \) is minimum admissible, i.e., it dominates all admissible functions:

Lemma 8 \( LS^u(x, t, r) \) is minimum admissible, i.e., \( LS^u(x, t, r) \) dominates any admissible sensitivity function \( \delta^u(x, t, r) \).

Even if \( LS^u(x, t, r) \) happens to be non-monotonic, we devote the next subsection to discuss that only a “weak”
monotonicity is enough for our mechanism. Also, we discuss other cases where the local dampening mechanism also performs well.

4.5 Relaxing monotonicity

We have shown theoretical guarantees for the accuracy of the shifted local dampening mechanism using stable functions. For sensitivity functions like the global sensitivity $\Delta u$ and the local sensitivity $LS^u(x, t)$, we have strong accuracy guarantees.

Strict monotonicity may be a complex goal to achieve. In the applications and datasets analyzed in our experimental section, none of them satisfy the strict monotonicity requirement. Yet, the shifted local dampening mechanism outperforms the exponential mechanism in our experiments.

For those sensitivity functions that violate monotonicity, we use the results in Sect. 4.4.1 as guide to construct a good sensitivity function. The same analysis also works here, Lemma 6 suggests that a sensitivity function should be as inclined as possible, i.e., a higher difference between two gaps $\alpha(x, t, r_i)$ and $\alpha(x, t, r_{i+1})$ implies higher accuracy. Also, the gaps $\alpha(x, t, r_i)$ should be as large as possible.

For the running example of this paper (Example 1), we designed an admissible function $\delta^{EBC}$ stated in Definition 20 for the use of the shifted local dampening mechanism. Figure 5b displays the value of $u(x, r)$ on the x-axis against $\delta^u(x, 0, r)$ for the Enron graph database [21]. This example clearly violates strict monotonicity. But it shows a weak monotonicity for the sensitivity function $\delta^{EBC}$ in the sense that $EBC(x, r)$ is still positively correlated with $\delta^{EBC}(x, t, r)$ with respect to $r$.

We argue that this kind of behavior is enough for a good performance of the shifted local dampening. Our empirical results corroborate with this argument.

Additionally, Lemma 6 also suggests that functions that do not exhibit correlation but have lower value than $\Delta u$ also perform well which is the case for local sensitivity $LS^u(x, t)$.

5 Application 1: Percentile selection

In this Section, we address the percentile selection problem. The goal is to output the label of the element with the closest value to the $p$-th percentile from a set of real numbers.

5.1 Problem statement

Given a database $x \in \mathbb{R}^n$ represented as a vector of real numbers $(x_1, \ldots, x_n)$. For simplicity’s sake, assume that every database $x$ is ordered such that $x_1 \leq \cdots \leq x_n$. Suppose that all the values lies in $[0, \Lambda]$, $0 \leq x_1 \leq \cdots \leq x_n \leq \Lambda$.

The task is to return the index $i$ where its element $x_i$ is as close as possible to the $p$-th percentile element $x_k$ where $k = \left\lceil \frac{pn+1}{100} \right\rceil$. Note that $\mathcal{R} = \{1, \ldots, n\}$. The utility function for a given index $i$ is the distance from $x_i$ to $x_k$ multiplied by $-1$ so that closer elements have higher utility score:

**Definition 18** (Utility function for percentile selection problem) $u_p(x, i) = -|x_k - x_i|$, where $k = \left\lceil \frac{pn+1}{100} \right\rceil$.

5.2 Related work

Nissim et al. [5] and McKenna et al. [20] have tackled reduced versions of this problem that work only with medians. Nissim et al. [5] tackled the numeric version of the median selection problem where the task is to return the median value itself and not the label of the median element. McKenna et al. [20] addressed binned version of the median selection problem where the data is binned in $k$ buckets and the goal is to return the median bin. The latter version has a low global sensitivity. However, our work discusses a more generic version that can return the label of elements of any percentile element in the dataset.

5.3 Private mechanism and sensitivity analysis

This problem is solved by a single call to a non-numeric mechanism. Here we use the exponential mechanism and the permute-and-flip mechanism to compare to the local dampening mechanism. The exponential mechanism and the permute-and-flip mechanism require the computation of the global sensitivity $\Delta u_p$, while the local dampening mechanism requires the computation of the element local sensitivity $u_p$.

5.3.1 Global sensitivity

The global sensitivity $\Delta u_p$ is set by the following scenario: let $p = 50$ implying that $k = \left\lceil \frac{n+1}{2} \right\rceil$. Let $x \in \mathbb{R}^n$, for $n \geq 2$, be a database where $x_1 = x_2 = \cdots = x_{n-1} = 0$ and $x_n = \Lambda$. Let $y = (y_1, \ldots, y_n) \in \mathbb{R}^n$ be a neighboring database of $x$ obtained from $x$ by changing the value of $x_n$ to $0$, $y_n = 0$. Thus we have that $y_1, \ldots, y_n = 0$. In what follows, $u_p(x, n) = \Lambda$ and $u_p(y, n) = 0$ as $x_k = y_k = 0$ which implies that $u(x, n) - u(y, n) = \Lambda$.

This happens to be the largest possible $|u_p(x, i) - u_p(y, i)|$ since $|u_p(x, i) - u_p(y, i)| \leq \Lambda$ for any $x, y \in \mathbb{R}^n$ and $i \in [1, n]$. The latter follows from the fact that the distance from $x_i$ to $x_k$ is positive and smaller than $\Lambda$, $0 \leq u_p(x, i) \leq \Lambda$ and $0 \leq u_p(y, i) \leq \Lambda$.

**Lemma 9** (Percentile Selection Global Sensitivity) $\Delta u_p = \Lambda$. 

 Springer
5.3.2 Element local sensitivity

Element local sensitivity at distance 0. Before calculating the element local sensitivity of \( u_p \) at distance \( t \), we discuss how to compute the element local sensitivity at distance 0 \( LS^{u_p}(x, 0, i) \).

Observe that a naive computation of \( LS^{u_p}(x, 0, i) \) is infeasible. It needs to iterate over each neighboring database \( y \) of \( x \) and take \( |u(x, i) - u(y, i)| \). The number of neighboring databases is infinite because we can set a given \( x_i \) to any real value in \([0, \Lambda]\).

\[
LS^{u_p}(x, 0, i) = \max_{y|d(x, y) \leq 1} |u_p(x, i) - u_p(y, i)|
\]

Thus we provide a way to efficiently compute \( LS^{u_p}(x, 0, i) \) in \( O(1) \) time complexity (Lemma 10).

**Lemma 10** (Percentile Selection Element Local Sensitivity at distance 0)

\[
LS^{u_p}(x, 0, i) = \max(|x_k - x_i|, x_{k+1} - x_k, x_k - x_{k-1}, p(x, i), q(x, i)),
\]

where

\[
p(x, i) = \begin{cases} 
\Lambda - x_i & \text{if } i > k \\
\Lambda - x_{k+1} & \text{if } i = k \\
\Lambda + x_i - 3x_k + x_{k+1} & \text{if } i < k 
\end{cases}
\]

\[
q(x, i) = \begin{cases} 
x_i & \text{if } i > k \\
x_{k-1} & \text{if } i = k \\
3x_k - x_i - x_{k-1} & \text{if } i < k 
\end{cases}
\]

where \( 0 \leq x_1 \leq \cdots \leq x_n \leq \Lambda \) and \( k = \left[ \frac{p_n+1}{100} \right] \).

Element local sensitivity at distance \( t \). Now we proceed to compute \( LS^{u_p}(x, t, r) \).

\[
LS^{u_p}(x, t, r) = \max_{y|d(x, y) \leq t} LS^{u_p}(y, 0, r)
\]

Given a distance \( t \), our task is to compute \( LS^{u_p}(y, 0, r) \) over all \( y \) such that \( d(x, y) \leq t \). A naive brute force approach would be infeasible since there are infinite databases at distance \( t \) from \( x \) as discussed previously for the computation of element local sensitivity at distance 0.

However, there is a small subset, referred as \( candidates(x, t, r) \), of \( \{y|d(x, y) \leq t\} \) where we can evaluate \( LS^{u_p}(y, 0, r) \) only on the databases of \( candidates(x, t, r) \). The databases \( \{y|d(x, y) \leq t\} - candidates(x, t, r) \) are safe to discard, i.e., there exists a database \( y \in candidates(x, t, r) \) where \( LS^{u_p}(x, t, r) = LS^{u_p}(y, 0, r) \). So we rewrite the element local sensitivity of \( u_p \) as:

**Lemma 11** (Element local sensitivity at distance \( t \) for percentile selection)

\[
LS^{u_p}(x, t, r) = \max_{candidates(x, t, r)} LS^{u_p}(y, 0, r).
\]

Algorithm 2 depicts how to compute \( candidates(x, t, r) \). The \( CopyAndReplace(x, i, v) \) function used in this algorithm makes a copy \( x' \) of the input database \( x \), sets the element \( x'_i \) to the value \( v \) and returns the ordered dataset \( x' \). The algorithm \( candidates(x, t, r) \) returns a subset of only 6 databases of \( \{y|d(x, y) \leq t\} \) that maximizes \( \max_{y|d(x, y) \leq t} LS^{u_p}(x, t, r) \). A sketch of the proof of Lemma 11 can be found in [22].

**Algorithm 2**: Candidates Algorithm

1. Procedure Candidates(Dataset \( x \), distance \( t \), range element \( r \))
2. if \( t = 0 \) then
3. \quad return \( (x) \)
4. end
5. \( k = \left[ \frac{p_n+1}{100} \right] \)
6. if \( t = 1 \) then
7. \quad \( x'_1 = CopyAndReplace(x, r, \Lambda) \)
8. \quad \( x'_2 = Copy(x) \)
9. \quad \( x'_3 = CopyAndReplace(x, r, 0) \)
10. \quad \( x'_4 = Copy(x) \)
11. \quad \( x'_5 = CopyAndReplace(x, k, \Lambda) \)
12. \quad \( x'_6 = CopyAndReplace(x, k, 0) \)
13. \quad return \( \{x'_1, x'_2, x'_3, x'_4, x'_5, x'_6\} \)
14. end
15. \( x_1, x_2, x_3, x_4, x_5, x_6 = Candidates(x, t - 1, r) \)
16. \( x'_1 = CopyAndReplace(x_1, k, 0) \)
17. \( x'_2 = CopyAndReplace(x_2, k, \Lambda) \)
18. \( x'_3 = CopyAndReplace(x_3, k, 0) \)
19. \( x'_4 = CopyAndReplace(x_4, k, \Lambda) \)
20. \( x'_5 = CopyAndReplace(x_5, k, 0) \)
21. \( x'_6 = CopyAndReplace(x_6, k, \Lambda) \)
22. return \( \{x'_1, x'_2, x'_3, x'_4, x'_5, x'_6\} \)

**Example 10**: (Algorithm 2 example run). In this example, we show a run of the Algorithm 2 using the following parameters: Dataset \( x = \{0, 1, 2, 3, 4, 5\} \), distance \( t = 2 \) and range element index \( r = 3 \). In this example, the datasets’ elements assume values in \([0, 5]\) (\( \Lambda = 5 \)) and \( p = 20 \).

Note that the call for Algorithm 2 for \( t = 2 \) has a recursive with \( t = 1 \). Thus we illustrate in Table 1 the datasets that Algorithm 2 produces for each call:
We tested most of the datasets from Hay et al. [23].

Datasets

5.4 Experimental evaluation

Datasets

The datasets returned by a call to Algorithm 2 when Table 1

Local dampening: differential privacy for non-numeric queries via local sensitivity

Table 1 The datasets returned by a call to Algorithm 2 when $x = \{0, 1, 2, 3, 4, 5\}$ and $r = 3$ for $t = 1$ and $t = 2$

| Datasets | $t = 1$                      | $t = 2$                      |
|----------|-----------------------------|-----------------------------|
| $x'_1$   | $\{0, 1, 2, 4, 5, 5\}$      | $\{0, 0, 1, 4, 5, 5\}$      |
| $x'_2$   | $\{0, 1, 2, 3, 4, 5\}$      | $\{0, 1, 1, 3, 4, 5, 5\}$  |
| $x'_3$   | $\{0, 0, 1, 2, 4, 5\}$      | $\{0, 0, 0, 2, 4, 5\}$      |
| $x'_4$   | $\{0, 1, 2, 3, 4, 5\}$      | $\{0, 1, 3, 4, 5, 5\}$      |
| $x'_5$   | $\{0, 1, 3, 4, 5, 5\}$      | $\{0, 0, 1, 4, 5, 5\}$      |
| $x'_6$   | $\{0, 0, 0, 1, 3, 4, 5\}$   | $\{0, 0, 0, 3, 4, 5, 5\}$   |

5.4 Experimental evaluation

Datasets

We tested most of the datasets from Hay et al. [23]. Our approach is specially accurate when the local sensitivity is far from the global sensitivity. We report the results for two datasets where the local sensitivity is reasonably smaller from the global sensitivity: PATENT dataset and HEPHTH dataset. Also we show one dataset where the local sensitivity is very close to the global sensitivity to show our approach behaves on this scenario: INCOME dataset.

Methods

We test three approaches for the percentile selection problem: (i) EMPercentileSelection, the exponential mechanism using global sensitivity; (ii) PFPercentileSelection, the permute-and-flip mechanism using the global sensitivity; and (iii) LDPercentileSelection, the local dampening mechanism using local sensitivity.

Evaluation

We measure the error: $\text{error} = \text{true}_\text{percentile}_\text{value} - \text{true}_\text{percentile}_\text{value}$. For the EMPercentileSelection and LDPercentileSelection methods, we report the mean error over 100, 000 runs. We vary $\epsilon \in [10^{-1}, 10^{2}]$ and $p = 50, 95, 99$.

Figure 6 displays the results. The results show that EMPercentileSelection and PFPercentileSelection mechanisms have very similar error in all the cases. For PATENT dataset, we observe up to 44%, 52%, 59%, for $p = 50, 90, 99$, respectively, of reduction in the error by the LDPercentileSelection in relation to both the EMPercentileSelection and PFPercentileSelection over the tested values for $\epsilon$. For HEPHTH dataset, the error reduction is at most 12%, 52%, 73%, for $p = 50, 90, 99$, respectively.

For the INCOME dataset, we observe up to 44%, 52%, 59%, for $p = 50, 95, 99$, respectively. We notice a slight reduction of error of at most 3% for $p = 99$. This behavior is similar to the other datasets from Hay et al. [23] not presented here where the local sensitivity is near to the global sensitivity.

6 Application 2: Influential node analysis

Identifying influential nodes in a network is an important task for social network marketing [24]. This analysis has great value for making a more effective marketing campaign since influential nodes have great capacity to diffuse a message through the network.

6.1 Problem statement

The influential node analysis problem is a query over an input graph database $G = (V, E)$ that releases the labels of $k$ nodes that maximize a given influence metric. In this work, we use the Egocentric Betweenness Centrality metric (EBC, Definition 19). Using EBC as an influence measure allows to identify influential nodes that are important in different loosely connected parties.

Definition 19 (Egocentric Betweenness Centrality (EBC) [12–14])

$$EBC(c) = \sum_{u,v \in N_c \mid u \neq v} \frac{p_{uv}(c)}{q_{uv}(c)},$$

where $N_c = \{v \in V \mid (c, v) \in E\}$ is the set of neighbors of the central node $c$, $q_{uv}(c)$ is the number of geodesic paths connecting $u$ and $v$ on the induced subgraph $G[N_c \cup \{c\}]$, and $p_{uv}(c)$ is the number of those paths that include $c$.

6.2 Private mechanism

We propose PrivTopk, a top-k algorithm template that chooses iteratively $k$ nodes that maximize EBC (Algorithm 3). In each iteration, the algorithm makes a call to a non-numeric mechanism (line 5) that returns a node that maximizes EBC that was not previously chosen.

We experiment with four instances of this algorithm template: (i) EMPrivTopk, where we replace line 5 with an exponential mechanism call; (ii) PFPrivTopk, where we replace line 5 with a permute-and-flip mechanism call; (iii) LDPrivTopk where we replace line 5 with a local dampening call; (iv) SLDDivTopk where we replace line 5 by a shifted local dampening mechanism.

Our algorithm issues $k$ calls to a private mechanism with privacy budget $\epsilon' = \epsilon/k$. By the sequential composition theorem (Theorem 1), the total privacy budget consumed in the entire algorithm is $\epsilon' \times k = (\epsilon/k) \times k = \epsilon$. Thus Algorithm 3 satisfies $\epsilon$-differential privacy.
Fig. 6 Expected error for the EMPercentileSelection and LDPercentileSelection methods and mean error over 100,000 runs for PFPercentileSelection where $\epsilon \in [10^{-1}, 10^2]$. X axis is in log scale.

### Algorithm 3: PrivTopk

1. Procedure PrivTopk(Graph $G = (V, E)$, Privacy Budget $\epsilon$, Integer $k$)
2. $\epsilon' = \epsilon / k$
3. $\Omega = \emptyset$
4. for $j \leftarrow 1$ to $k$
5.   $v = \text{MEC}(G, \epsilon', EBC, V)$ // Non-numeric mechanism call
6.   $\Omega = \Omega \cup \{v\}$
7. end
8. return $\Omega$

### 6.3 Sensitivity analysis

**Global sensitivity** We need to provide the global sensitivity for EBC to the Exponential Mechanism and the permute-and-flip mechanism:

**Lemma 12** (EBC global sensitivity) The global sensitivity $\Delta EBC$ for EBC is given by

$$\Delta EBC = \max \left( \frac{\Delta(G)(\Delta(G) - 1)}{4}, \Delta(G) \right),$$

where $\Delta(G)$ is the maximum degree of the input graph $G$. 
In this work, we assume the maximum degree is a public information or that we have an upper bound for it.

**Element local sensitivity** For the local dampening call, we provide an upper bound to the element local sensitivity using the sensitivity function \( \delta^{\text{EBC}} \).

**Definition 20** (Sensitivity function \( \delta^{\text{EBC}}(G, t, v) \)). The sensitivity function \( \delta^{\text{EBC}} \) for EBC is defined as

\[
\delta^{\text{EBC}}(G, t, v) = \max \left( \frac{d^G(v)+t}{4}, d^G(v)+t \right),
\]

where \( d^G(v) \) denotes the degree of \( v \) in \( G \), i.e., \( d^G(v) = \left| N^G_v \right| \).

We also show that \( \delta^{\text{EBC}}(G, t, v) \) is admissible (Lemma 13).

**Lemma 13** The sensitivity function \( \delta^{\text{EBC}}(G, t, v) \) is admissible.

In terms of correlation between \( EBC(v) \) and \( \deg^G(v)+t \), note that for a given node \( v \) with degree \( d^G(v) \), there are \( \frac{d^G(v)}{2} \) terms in the \( EBC \) equation for \( v \) (Definition 19), i.e., pairs \( (u, z) \in N^G_v \). As each term contributes at most 1 to \( EBC \), it suggests that there is a correlation between \( EBC(v) \) and \( \deg^G(v)+t \) and consequently, between \( EBC(v) \) and \( \delta^{\text{EBC}}(G, t, v) \). Empirical observation of the datasets confirmed that correlation. For this reason, the shifted local dampening mechanism call in SLDPrivTopk is suitable.

### 6.4 Related work

The literature provides some work to release statistics on graphs which are presented in this section. Also, we present some work on releasing linear statistics over relational databases that can be used to release EBC and compare experimentally to our approach.

#### 6.4.1 Differentially private graph analysis

Kasiviswanathan et al. [8] observed that the sensitivity of many graph problems is a function of the maximum degree of the input graph \( G \), so they proposed a generic projection that truncates the maximum degree of \( G \). This projection is built upon smooth sensitivity framework [5] but the target query is answered using the global sensitivity on the truncated graphs which may still be high. Moreover, it satisfies a weaker definition of privacy: \((\epsilon, \delta)\)-differential privacy.

There are a number of works that aims to publish subgraph counting as k-triangles, k-stars and k-cliques. Kasiviswanathan et al. [8] also address this problem applying a linear programming-based approach that releases those counts for graphs that satisfy \( \alpha \)-decay.

A new notion of sensitivity called restricted sensitivity was introduced by Blocki et al. [6] to answer subgraph counts. In this work, the querier has some beliefs about the structure of the input graph, so the restricted sensitivity measures sensitivity only on the subset of graphs which are believed to be inputs to the algorithm. However, this work satisfies only \((\epsilon, \delta)\)-differential privacy.

Blocki et al. [6] introduced the ladder functions to answer subgraph counts. A latter function is a structure built upon the local sensitivity of the subgraph count. It ranks the possible outputs of the subgraph count query and samples a given output using the exponential mechanism with low sensitivity.

The work presented in [7] is an application of the smooth sensitivity framework [5] for answering to subgraph count queries. The authors provide the bounds of the local sensitivity of k-triangles and k-stars and show that they are more accurate than related work.

About centrality metrics. A recent work by Laeuchli et al. [25] analyzes three centrality measures on graphs: eigenvector, Laplacian and closeness centrality. The result is that releasing those metrics using either the Laplace mechanism (based on global sensitivity) or the smooth sensitivity framework (based on local sensitivity) is infeasible. To show that, they demonstrate that the local sensitivity is unbounded or, even it is bounded, it is too large and it results in overwhelming addition of noise.

To the best of our knowledge, in the literature, none of the works on graph analysis tackles top-k or EBC release.

#### 6.4.2 Releasing linear statistics over relational databases

A body of work is available in the literature on answer linear queries, i.e., queries that can be answered by linear aggregation, over relational databases using SQL under differential privacy.

As will be shown in Sect. 6.5.1, the EBC metric can be computed by issuing a set of count SQL queries with cyclic joins and GROUP BY clauses over a graph stored in relational database. Thus we survey works that can possibly answer to this kind of query.

Local sensitivity has been used to answer full acyclic join queries [26]. This approach lacks generality as it cannot compute SQL queries with cyclic joins and GROUP BY clauses for which it is required for EBC queries. The recursive mechanism [27] can answer linear queries with unrestricted joins with GROUP BY clauses; however, it requires the target function \( f \) to be monotonic, i.e., inserting a new individual in the database always causes \( f \) to increase (or always decrease). This condition is not satisfied by EBC.

PrivateSQL [28] is an approach that can answer linear queries with cyclic joins and correlated subqueries with GROUPBY clauses.
This approach requires a representative workload $Q$ as input, a primary relation in the database. The representative workload is used in the VSelector module to identify a set of views over the base relations that support the analyst queries and then generates a set of views that can answer all the analyst’s queries. The VRewrite module rewrites using truncation operators and semijoin operators to bound the sensitivity. Therefore, the sensitivity for each query is computed in the SensCalc module and a fraction of the budget is given to each query in the BudgetAllocator module. The PrivSynGen generates a synopsis for each view. A synopsis is a compact representation of a view that captures important information from it and can approximate the answer of a given query on the view. The private synopses are released under differential privacy, and queries can be executed over a given query on the view. The private synopses are released under differential privacy, and queries can be executed over them without using privacy budget. The synopsis generation is based on non-negative least squares inference $[29]$. The sensitivity computation for each view is based on Flex $[30]$ augmented with truncation operators.

Then when a new query is issued to the system, this query is rewritten as linear queries over the private views’ synopses in the MapQuery module. Since graph databases can be modeled as a table Node(id) and a table Edge(source, target), we carry out an experimental evaluation in Sect. 6.5.1.

### 6.5 Experimental evaluation

**Datasets** We use three real-world graph datasets: (1) Enron is a network of email communication obtained from around half million emails. Each node is an email address and an edge connects a pair of email addresses that exchanges emails ($|V| = 36, 692, |E| = 183, 831$ and $\Delta(G) = 1, 383$); (2) DBLP is a co-authorship network where two authors (nodes) are connected if they published at least one paper together ($|V| = 317, 080, |E| = 1, 049, 866$ and $\Delta(G) = 343$); (3) Github is a network of developers with at least 10 stars on the platform. Developers are represented as nodes and an edge indicates that two developers follow each other ($|V| = 37, 700, |E| = 289, 003$ and $\Delta(G) = 9, 458$). $V$ is the set of vertices, $E$ is the set of edges, and $\Delta(G)$ is the maximum degree of a graph $G$. All datasets can be found on Stanford Network Dataset Collection $[21]$.

**Methods** We compare the four versions of PrivTopk (algorithm 3): (1) EMPPrivTopk, using the exponential mechanism, (2) PFPrivTopk, using permute-and-flip mechanism, (3) LDPrivTopk using local dampening mechanism and (4) SLDPrivTopk using shifted local dampening mechanism.

**Evaluation** We evaluate the accuracy by the percentage of common nodes to the retrieved top-$k$ set and the true top-$k$ set, i.e., $(|\text{retrieved}_\text{topk} \cap \text{true}_\text{topk}|)/k$. We report the mean accuracy in 100 simulations. We set $k \in \{5, 10, 20\}$ and a range for privacy budget $\epsilon \in [10^{-3}, 10^4]$.

Figure 7 displays the results. We first note that the global sensitivity-based approaches, EMPPrivTopk and PFPrivTopk, exhibit low accuracy for low values of $\epsilon$. This is due to the high global sensitivity: $\Delta EBC = 22, 361, 076.5$ for github dataset, $\Delta EBC = 477, 826.5$ for DBLP dataset and $\Delta EBC = 29, 326.5$ for Enron dataset. Also, the LDPrivTopk algorithm suffers from the inversion problem (Sect. 4.1) while SLDPrivTopk could exploit the correlation between $EBC$ and $\delta EBC$ to fix this problem.

We observe a clear pattern where the methods perform worse as $k$ grows. This is explained by the fact that each call to the non-numeric mechanism uses $\epsilon/k$ of the total privacy budget $\epsilon$ (Algorithm 3). Thus, larger $k$ implies that less of the privacy budget is used in each non-numeric mechanism call which hurts accuracy.

For SLDPrivTopk, we note that we need different level of privacy budget for each dataset reasonable accuracy. This is explained by a number of factors. For Github dataset, the distribution of EBC is heavy tailed and thus the nodes with high EBC have a higher probability to be correctly picked with low privacy budget. On the other hand, for the DBLP dataset we need more privacy budget as it has roughly 10 times more nodes than the other datasets, which dilute the probability of the nodes with higher EBC.

Our approach SLDPrivTopk achieves the same level of accuracy with privacy values 3 to 4 orders of magnitude less than EMPPrivTopk and 2 to 3 orders of magnitude less than PFPrivTopk.

#### 6.5.1 Comparison to PrivateSQL

We carry out an experimental comparison of local dampening mechanism to the PrivateSQL on Influential Node Analysis problem. For this application, PrivateSQL is not particularly scalable (as discussed further), so we performed the experiments with smaller datasets and test with fewer values of privacy budgets.

PrivateSQL addresses the Influential Node Analysis problem by computing the counts $q_{uv}(c)$ and $p_{uv}(c)$ for all $u, v \in N_c$ (Definition 11) to compute EBC and takes the top-$k$ nodes with highest EBC score. Note that it considers only for the terms $p_{uv}(c)/q_{uv}(c)$ where the distance from $u$ to $v$ in $G[N_c \cup \{c\}]$ is 2 which is the maximum possible distance as $u, v \in N_c$. If their distance is 1 (i.e., $u$ and $v$ are neighbors), the term $p_{uv}(c)$ is 0 since the geodesic path of length 1 from $u$ to $v$ ($u, v \neq c$) cannot contain $c$. Thus, for PrivateSQL, we pose private queries $Q(u, v, c)$ that returns (1) 1 if $u$ and $v$ are neighbors, (2) 0 if the distance from $u$ to $v$ is larger than 2 and (3) $q_{uv}(c)$, otherwise.

Therefore, when $Q(u, v, c)$ is not equal to 0, it means that the term $p_{uv}(c)/q_{uv}(c)$ should be accounted in the EBC...
For every \( u \) which are obtained by choosing a node sample \( S \), reason, we perform experiments with samples \( EBC \) from noisy definition. In that case, we obtain a noisy estimate for \( q_{uv}(c) \) from \( Q(u, v, c) \). A noisy estimate for \( p_{uv}(c) \) can be derived from noisy \( q_{uv}(c) \) by setting \( p_{uv} \) to 0 if \( q_{uv}(c) = 0 \) or to 1 if \( q_{uv}(c) > 0 \). The rationale is that exactly one of the paths of length 2 from \( u \) to \( v \) contains \( c \) as a subpath.

The set \( N_c \) is itself private information. Hence, to compute EBC for every \( c \in V(G) \) we need to compute \( Q(u, v, c) \) for every \( u, v \in V(G) \). This results in a total number of \( O(n^3) \) queries which poses a scalability problem. For this reason, we perform experiments with samples \( S \) of the graphs which are obtained by choosing a node sample \( S_0 \) in breadth-first search fashion with a random seed node and then we set \( S = G[S_0] \).

Table 2 displays the mean accuracy for 10 runs on 10 sample graphs with \( |S_0| = 50 \) nodes with \( k \in \{1, 2, 3\} \) for each \( \epsilon \in \{0.1, 0.5, 1.0, 5.0, 10.0\} \). We compare the best local dampening based algorithm SLDPrivTopk (PTK) with the PrivateSQL-based approach (PSQL).

PrivateSQL approach generated one private view for each node in the graph. Thus, the privacy budget needs to be divided by the number of nodes \( n \) which implies that accuracy is hurt as \( n \) grows. Moreover, the sensitivity for each view is high, e.g., sensitivity is 1448 when \( \Delta(G) = 10 \). This entails in a poor performance for the PrivateSQL-based approach.

### 6.5.2 Scalability analysis

To test the scalability of our approach, we measure the execution time of the best approach presented in Sect. 6.5, the SLDPrivTopk algorithm.

The main factor impacting on the execution time is the size of the graph. Thus, we use DBLP dataset which is the largest graph used in Sect. 6.5 and we extract subgraphs of this graph of increasing sizes to test against SLDPrivTopk algorithm.

Specifically, given a graph \( G = (V, E) \), let \( n' \in \{1, \ldots, n\} \) be the size of the subgraph to be extracted where \( n = |V| \). We generate a random sample \( S \) of \( V \) of size \( n' \) and we extract the induced graph \( G[S] \) on \( S \).

Thus, in this experiment, we set \( n' \in \{0.05n, 0.1n, \ldots, 0.95n, n\} \). For each \( n' \), we extract 10 subgraphs of size \( n' \) as described previously, execute the SLDPrivTopk algorithm and measure the mean runtime in seconds. We executed this in an Intel(R) Core(TM) i7-7820X CPU @ 3.60GHz CPU, 128GB RAM and Ubuntu 20.04. The SLDPrivTopk algorithm was implemented using Python 3.8/Cython. The results are reported in Fig. 5c.

The time complexity of the SLDPrivTopk algorithm cannot be described in a simple closed form. Thus, we empirically assess how runtime behaves as a function of the size of the dataset by fitting a function of the form \( n'\gamma + c = time \) in the experiment data, where \( n \) is the size of the subgraph and time is the runtime in seconds of SLDPrivTopk algorithm. We estimate \( \gamma \) and \( c \) using the least squares method and we vary \( \gamma \in \{1, 4\} \). The \( \gamma \) that produced the minimum squared error is \( \gamma = 1.9 \).
7 Application 3: ID3 decision tree induction

Classification based on decision tree is an important tool for data mining [31]. Specifically, decision trees are a set of rules that are applied to the input attributes to decide to which class a given instance belongs.

Creating a decision tree manually is a burden. Thus many approaches for automatically building decision trees were proposed. One of the most known tree induction algorithms is the ID3 algorithm [32] that receives a dataset and outputs a decision tree.

The ID3 algorithm [32] starts with the root node containing the original set. Then the algorithm greedily chooses an unused attribute to split the set and generate child nodes. The selection criterion is Information Gain (IG), given by

\[ IG(A) = \sum_{j \in C} \left( \frac{\tau^T_j}{\tau^T} \log \frac{\tau^T_j}{\tau^T} \right) - \sum_{i \neq j} \left( \frac{\tau^T_i}{\tau^T} \log \frac{\tau^T_i}{\tau^T} \right) \]

This process continues recursively for the child node until splitting does not reduce entropy or the maximum depth is reached.

7.1 Problem statement

A decision tree induction algorithm takes as input a dataset \( T \) with attributes \( A = \{A_1, \ldots, A_d\} \) and a class attribute \( C \) and produces a decision tree. The task is to build a decision tree in a differentially private manner. Specifically, we base our approach in one of the most known tree induction algorithms, the ID3 algorithm.

The notation for this section is summarized in Table 3. All logarithms are in base 2. When it is clear from the context, we drop the superscript \( T \) from the notations.

7.2 Related work

Blum et al. [33] presented the SuLQ framework that provides primitives for data mining algorithms. As an application, they introduced a differentially private adaptation for ID3 algorithm where it computes the information gain based on the noisy counts provided by SuLQ. This approach applies two operators from SuLQ: (1) NoisyCount that uses Laplace mechanism to return private estimate of a count query, and (2) Partition that splits the dataset in disjoint subsets so that the privacy budgets for the queries over each subset do not sum up (parallel composition, Theorem 2) meaning that we can make a more efficient use of the privacy budget. A major drawback of this algorithm is that it requires to query the noisy counts (via NoisyCounts) for each attribute, so the privacy budget needs to be split to those queries. This entails in a small budget per query and in a larger noise magnitude.

To overcome this, Friedman et al. [34] introduced a variation of the SuLQ algorithm (Algorithm 4). This algorithm replaces the many NoisyCount calls for a single call to the exponential mechanism. Line 12 (Algorithm 4) is the call for the exponential mechanism that was previously several calls to NoisyCount.

The procedure Build_DiffPID3 in Algorithm 4 starts by checking for the construction of a leaf node (Line 5-7). It verifies if there the dataset has any remaining attribute and if there is enough instances to make new splits. In lines 8-10, it partions the data based on the class attribute \( C \), and it privately queries the count of each partition and creates a leaf with the class label of the largest partition. The Lines 12-15 creates a new decision rule recursively. It starts by privately choosing the attribute with largest IG using the exponential mechanism, and then it partitions the dataset using the chosen attribute and calls Build_DiffPID3 with each partition to create the sub-trees.

To the best of our knowledge, our work is the first to apply local sensitivity to single trees which was an open question pointed out in a recent survey on private decision trees [35].

Many other works address the private construction of random forests [36–41]. Interestingly, local sensitivity was used for building random forests [37,38] using smooth sensitivity. This shows a promising future direction of our work which is applying local dampening to construct random forests.
Algorithm 4: GlobalDiffPID3

1. Procedure GlobalDiffPID3(Dataset \( T \), Attribute Set \( A \), Class attribute \( C \), Depth \( d \), Privacy Budget \( \epsilon \))
2. \( \epsilon' = \epsilon/(2(d + 1)) \)
3. return Build_DiffPID3\( (T, A, C, d, \epsilon') \)

4. Procedure Build_DiffPID3\( (T, A, C, d, \epsilon) \)
5. \( t = \max_{A \in A} |A| \)
6. \( \bar{N}_T = \text{NoisyCount}_t(T) \)
7. if \( A = \emptyset \) or \( d = 0 \) or \( \bar{N}_{Tc} < \frac{T}{\epsilon} \) then
8. \( T_c = \text{Partition}(T, \forall c \in C : r_c = c) \)
9. \( \forall c \in C : N_c = \text{NoisyCount}_t(T_c) \)
10. return a leaf labeled with \( \arg \max_c(N_c) \)
11. end
12. \( \bar{A} = \mathcal{E}(T, \epsilon, IG, A) // \text{Exp. mechanism call} \)
13. \( T_i = \text{Partition}(T, \forall i \in \bar{A} : r_{Ai} = i) \)
14. \( \forall i \in \bar{A} : \text{Subtree}_i = \text{Build_DiffPID3}((T_i, A \setminus \bar{A}, C, d - 1, \epsilon) \)
15. return a tree with a root node labeled \( \bar{A} \) and edges labeled 1 to \( \bar{A} \) each going to \( \text{Subtree}_i \)

7.3 Private mechanism

We use the algorithm GlobalDiffPID3 \cite{34} (Algorithm 4) as a template. We aim to adapt it for the use of the local dampening mechanism and to the shifted local dampening mechanism producing the LocalDiffPID3 and ShiftedLocalDiffPID3, respectively. In the following, we make a discussion about the split criterion, the global sensitivity of the split criterion for the exponential mechanism and the element local sensitivity for the local dampening.

Split criterion. In this work, we address one of the most traditional split criterion, information gain (IG). It is given by the entropy of the class attribute \( C \) in \( T \) minus the obtained entropy of \( C \) splitting the tuples according to an attribute \( A \in A \).

\[ IG(T, A) = H_C(T) - H_{C|A}(T), \]

where \( H_C(T) \) is entropy with respect to the class attribute \( C \):

\[ H_C(T) = -\sum_{c \in C} \frac{\tau_c}{\tau} \log \frac{\tau_c}{\tau}, \]

and \( H_{C|A}(T) \) is the entropy obtained by splitting the instances according to attribute \( A \):

\[ H_{C|A}(T) = \sum_{j \in A} \frac{\tau_j}{\tau} . H_C(T_j^A). \]

Since \( H_C(T) \) does not depend on \( A \), we can further simplify the utility function IG:

\[ IG(T, A) = -\tau \cdot H_{C|A}(T) \]

\[ = -\sum_{j \in A} \sum_{c \in C} \tau_j^A \cdot \log \left( \frac{\tau_{j,c}}{\tau_j^A} \right). \]

Global sensitivity The exponential mechanism requires the computation of the global sensitivity for IG. It is given by \( \Delta IG = \log(N + 1) + 1/\ln 2 \) \cite{34} where \( N \) is the size of the dataset \( T \). The global sensitivity case can be achieved by \( T \) and \( T' \) where:

1. \( T \) has all tuples with values for \( A \) equal to a single value \( j \in A \) and all tuples class attribute \( C \) are set to a value different from a given value \( c \in C \) (i.e., \( \tau_j^A = \tau \) and \( \tau_j^{A,c} = 0 \));
2. \( T' \) is obtained from \( T \) by adding a tuple \( r \) where \( r_A = j \) and \( r_C = c \).

Element local sensitivity In our experiments, we observed that this mentioned case for the global sensitivity is not frequent in real datasets. For those datasets, a local measurement of the sensitivity can be about one order of magnitude lower than the global sensitivity. To this matter, we replace line 12 of Algorithm 4 for a local dampening mechanism call producing the algorithm LocalDiffPID3.

Element Local Sensitivity at distance 0 To use the local dampening mechanism, we provide means to efficiently compute the element local sensitivity for IG (Lemma 15). The element local sensitivity at distance \( t \) measures \( LSI_{IG}(T', 0, A) \) for all datasets \( T' \) such that \( d(T, T') \leq t \).

We first show how to obtain \( LSI_{IG}(T', 0, A) \):

Lemma 14 (Element local sensitivity at distance 0 for IG)

Given a dataset \( T \) and the attribute set \( A \), \( LSI_{IG}(T, 0, A) \) produces the element local sensitivity for IG at distance 0:

\[ LSI_{IG}(T, 0, A) = \max_{j \in A, c \in C} h(\tau_{j,T}^A, \tau_{j,c,T}), \]

where

\[ h(a, b) = \max(f(a) - f(b), g(b) - g(a)), \]

\[ g(x) = x . \log((x + 1)/x) - \log(x + 1), \]

\[ f(x) = x . \log((x + 1)/x) + \log(x + 1). \]

Assume that \( g(x) = 0 \) for \( x \leq 1 \) and \( f(x) = 0 \) for \( x \leq 0 \). Note that, the expression \( g(\tau_{j,c,T}^A) - g(\tau_{j,c,T}^{A,T}) \) measures the impact of the removal of a tuple \( r \) such that \( r_A = j \) and \( r_C = c \) and the expression \( f(\tau_{j,c,T}^A) - f(\tau_{j,c,T}^{A,T}) \) measures the addition of tuple \( r \). Thus to obtain \( LSI_{IG}(T, 0, A) \), we need to measure, for all \( j \in A \) and \( c \in C \), the addition or removal of the tuple \( r \) where \( r_A = j \) and \( r_C = c \), i.e., \( h(\tau_{j,c,T}^{A,T} - \tau_{j,c,T}^A, \tau_{j,c,T}^{A,T} - \tau_{j,c,T}^A) = \max(f(\tau_{j,T}^{A,T} - \tau_{j,T}^A), g(\tau_{j,c,T}^{A,T} - \tau_{j,c,T}^A)). \)

Element local sensitivity at distance \( t \) We use a similar idea to compute \( LSI_{IG}(T, t, A) \). \( LSI_{IG}(T, t, A) \) searches for the
largest $LS^u(T', 0, A)$ over all datasets $T'$ where $d(T, T') \leq t$:

$$LS^I_G(T, t, A) = \max_{T'} \max_{T'|d(T, T') \leq t} LS^u(T', 0, A) = \max_{c \in C} \max_{j \in A} h(\tau^A_j, \tau_j^r, \tau^A_j).$$

Exhaustively iterating over all $T'$ to compute $h(\tau^A_j, \tau_j^r, \tau^A_j)$ is not feasible since the number of datasets $T'$ grows exponentially with respect to $t$. However, we can restrict the number of evaluations of $h$ by discarding some of the datasets $T'$.

To this end, we introduce the algorithm $Candidates$ $(T, t, j, c)$ (Algorithm 5) that produces a subset of the set of the pairs $(\tau^A_j, \tau_j^r)$ of all datasets $T'$ such that $d(T, T') = t$, i.e., $Candidates(T, t, j, c) \subseteq \{(\tau^A_j, \tau_j^r) | d(T, T') = t\}$.

**Algorithm 5: Candidates Algorithm**

```
1 Procedure Candidates(Dataset T, distance t, attribute value j, class attribute value c)
2    if t = 0 then
3        return \{(\tau^A_j, \tau_j^r)\}
4    end
5    candidates = \{\}
6    for each pair (a, b) \in Candidates(T, t - 1, j, c) do
7        if a > 0 and b > 0 then
8            candidates = candidates \cup \{(a - 1, b - 1)\}
9        end
10       if a < t then
11           candidates = candidates \cup \{(a + 1, b)\}
12       end
13    end
14    return candidates
```

The Candidates algorithm has two important properties:

1. $Candidates(T, t, j, c)$ contains the pair $(\tau^A_j, \tau_j^r)$ such that $h(\tau^A_j, \tau_j^r)$ is maximum, i.e., $h(\tau^A_j, \tau_j^r) = \max_{T'|d(T, T') = t} h(\tau^A_j, \tau_j^r)$ (Lemma 15);
2. It is memorizable, when computing $D_{IG, \delta^G}$, we evaluate $LS^I_G(T', t, A)$ several times in increasing order of $t$ then one can store the result of the call to $Candidates(T, t - 1, j, c)$ to execute $Candidates(T, t, j, c)$ (line 6) efficiently.

Thus $LS^I_G(T, t, A)$ is given by:

$$LS^I_G(T, t, A) = \max_{j \in A} \max_{c \in C} h(a, b), (a, b) \in Candidates(T', t, j, c) \text{ for } 0 \leq t' \leq t.$$ In turn, the computation of $LS^I_G(T, t, A)$ is also memoizable. One can store a previous call to $LS^I_G(T, t - 1, A)$ to compute $LS^I_G(T, t, A)$ as:

$$LS^I_G(T, t, A) = \max_{j \in A} \max_{c \in C} h(a, b), (a, b) \in Candidates(T, t, j, c) \text{ for } 0 \leq t' \leq t.$$ In the datasets used in our experiments, $LS^I_G$ and $IG$ show correlation. Consequently, we also replace the exponential mechanism call on line 12 in algorithm 4 by a call to the Shifted local dampening with $LS^I_G$, which produces the ShiftedLocalDiffPID3.

**Continuous attributes support** An important feature introduced in C4.5 algorithm [42] is the support for continuous attributes. To support continuous attributes, we use a simpler approach that performed well in our experiments and it is also used in [34]. We discretize the continuous attributes into $b$ evenly spaced bins on the dataset and use them as discrete attributes.

### 7.4 Experimental evaluation

**Datasets** We use of three tabular datasets: (1) National Long Term Care Survey (NLCTS) [43] is a dataset that contains 16 binary attributes of 21,574 individuals that participated in the survey, (2) American Community Surveys (ACS) dataset [44] includes the information of 47,461 rows with 23 binary attributes obtained from 2013 and 2014 ACS sample sets in IPUMS-USA and (3) Adult dataset [45] contains 45,222 records (excluding records with missing values) with 12 attributes where 8 are discrete and 4 are continuous.

**Methods** We compare the three versions of the DiffPID3 (algorithm 4): (1) GlobalDiffPID3 using the exponential mechanism, (2) LocalDiffPID3 using the local dampening mechanism and (3) ShiftedLocalDiffPID3 using the local dampening mechanism with shifting.

**Evaluation** We evaluate the accuracy of the approach by reporting the mean accuracy across the 10 runs of a ten-fold cross validation. We set typical values for depth and $\epsilon$: depth $\in \{2, 5\}$ and $\epsilon \in \{0.01, 0.05, 0.1, 0.5, 1.0, 2.0\}$. Figure 8 presents the results. We observe that the LocalDiffPID3 improves on the GlobalDiffPID3 in almost every privacy budget value, up to 5%, while ShiftedLocalDiffPID3
improves a little more in relation to the LocalDiffPID3, up to 1%.

For the ACS dataset, the inversion problem (Sect. 4.1) appears. Specifically, for depth = 2, the second and the third attributes with largest IG become the third and second attributes, respectively, with larger Dampered IG. As a consequence, as \( \epsilon \) grows, LocalDiffPID3 tends to pick the first and the third attributes with largest Information which is suboptimal. ShiftedLocalDiffPID3 is less prone to suffer from the inversion problem in larger depths, i.e., depth = 5, since it can pick, in deeper levels, those attributes that loose rank (see Fig. 8f). ShiftedLocalDiffPID3 improves at most 8% on GlobalDiffPID3.

For the Adult Dataset, the LocalDiffPID3 improves a little over the GlobalDiffPID3. However, ShiftedLocalDiffPID3 improves over GlobalDiffPID3 up to 4%.

8 Conclusion

In this paper, we introduced the Local Dampening mechanism, a novel framework to provide Differential Privacy for non-numeric queries using local sensitivity. We have shown that using local sensitivity on non-numeric queries reduces the magnitude of the noise added to achieve Differential Privacy which makes the answer of those queries more useful. We evaluated our approach on three applications: (1) Percentile Selection problem that our approach outperformed global sensitivity-based approaches; (2) Influential node analysis which benefited greatly from the use of local sensitivity; and (3) Decision Tree induction which improves on approaches that use the exponential mechanism for this task based on Information Gain.

Our paper has laid the foundations for providing DP for non-numeric queries using local sensitivity. We have achieving a deeper theoretical understanding of the Local Dampening mechanism to understand the class of problems for which it can provide significant gains over the Exponential mechanism. There are many interesting directions of future work. First, as discussed in Sect. 7, any problem in the literature that has used the Exponential mechanism for non-numeric queries to guarantee DP is a candidate problem that could potentially benefit from using our local dampening mechanism instead, and worthy of future work. Second, it would be interesting to tackle other graph influence/centrality metrics for Influential Node analysis, such as PageRank. Third, applying the local dampening mechanism for private evolutionary algorithms is a promising future direction.

References

1. Commission, E.: 2018 reform of EU data protection rules (2018). https://gdpr-info.eu/
2. Brasil: Lei geral de proteção de dados pessoais (LGPD) (2018). http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm
3. Dwork, C.: Differential privacy. Encyclopedia of Cryptography and Security pp. 338–340 (2011)
4. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Theory of Cryptography Conference, pp. 265–284. Springer, Berlin (2006)
5. Nissim, K., Raskhodnikova, S., Smith, A.: Smooth sensitivity and sampling in private data analysis. In: Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, pp. 75–84. ACM (2007)
6. Blocki, J., Blum, A., Datta, A., Sheffet, O.: Differentially private data analysis of social networks via restricted sensitivity. In: Proceedings of the 4th Conference on Innovations in Theoretical Computer Science, pp. 87–96. ACM (2013)
7. Karwa, V., Raskhodnikova, S., Smith, A., Yaroslavtsev, G.: Private analysis of graph structure. PVLDB 4(11), 1146–1157 (2011)
8. Kasiviswanathan, S.P., Nissim, K., Raskhodnikova, S., Smith, A.: Analyzing graphs with node differential privacy. In: Theory of Cryptography Conference, pp. 457–476. Springer, Berlin (2013).
9. Lu, W., Miklau, G.: Exponential random graph estimation under differential privacy. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 921–930. ACM (2014)

10. Zhang, J., Cormode, G., Procopiuc, C.M., Srivastava, D., Xiao, X.: Private release of graph statistics using ladder functions. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 731–745. ACM (2015)

11. McSherry, F., Talwar, K.: Mechanism design via differential privacy. In: 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07), pp. 94–103 (2007). https://doi.org/10.1109/FOCS.2007.66

12. Freeman, L.C.: Centrality in social networks conceptual clarification. Soc. Netw. 1(3), 215–239 (1978)

13. Marsden, P.V.: Egocentric and sociocentric measures of network centrality. Soc. Netw. 24(4), 407–422 (2002)

14. Everett, M., Borgatti, S.P.: Ego network betweenness. Soc. Netw. 27(1), 31–38 (2005)

15. de Farias, V.A.E., Brito, F.T., Flynn, C., Machado, J.C., Majumdar, S., Srivastava, D.: Local dampening: differential privacy for non-numeric queries via local sensitivity. CoRR (2020). arXiv:2012.04117

16. Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., Naor, M.: Our data, ourselves: privacy via distributed noise generation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 486–503. Springer, Berlin (2006)

17. McSherry, F.D.: Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, pp. 19–30 (2009)

18. Machanavajjhala, A., He, X., Hay, M.: Differential privacy in the wild: a tutorial on current practices and pen challenges. In: Proceedings of the 2017 ACM SIGMOD International Conference on Management of data, pp. 1727–1730. ACM (2017)

19. Farias, V.A., Brito, F.T., Flynn, C., Machado, J.C., Majumdar, S., Srivastava, D.: Local dampening: differential privacy for non-numeric queries via local sensitivity. PVLDB 14(4), 521–533 (2020)

20. McKenna, R., Sheldon, D.R.: Permute-and-flip: a new mechanism for differentially private selection. In: Advances in Neural Information Processing Systems, vol. 33 (2020)

21. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data (2014)

22. Farias, V.: Local Dampening: Differential Privacy for Non-numeric Queries via Local Sensitivity. Ph.D. thesis, Universidade Federal do Ceará (2021). https://repositorio.ufc.br/handle/riufc/59462

23. Hay, M., Machanavajjhala, A., Miklau, G., Chen, Y., Zhang, D.: Principled evaluation of differentially private algorithms using dpbench. In: Proceedings of the 2016 International Conference on Management of Data, SIGMOD’16, pp. 139–154. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2882903.2882931

24. Ma, H., Yang, H., Lyu, M.R., King, I.: Mining social networks using heat diffusion processes for marketing candidates selection. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management, pp. 233–242 (2008)

25. Lauechli, J., Ramirez-Cruz, Y., Trujillo-Rasua, R.: Analysis of centrality measures under differential privacy models. arXiv preprint arXiv:2103.03556 (2021)

26. Tao, Y., He, X., Machanavajjhala, A., Roy, S.: Computing local sensitivities of counting queries with joins. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 479–494 (2020)

27. Chen, S., Zhou, S.: Recursive mechanism: towards node differential privacy and unrestricted joins. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 653–664 (2013)

28. Kotsogiannis, I., Tao, Y., He, X., Fananeepour, M., Machanavajjhala, A., Hay, M., Miklau, G.: PrivateSQL: a differentially private SQL query engine. PVLDB 12(11), 1371–1384 (2019)

29. Li, C., Miklau, G., Hay, M., McGregor, A., Rastogi, V.: The matrix mechanism: optimizing linear counting queries under differential privacy. The VLDB journal 24(6), 757–781 (2015)

30. Johnson, N., Near, J.P., Song, D.: Towards practical differential privacy for SQL queries. PVLDB 11(5), 526–539 (2018)

31. Kotsiantis, S.B., Zaharakis, I., Pintelas, P.: Supervised machine learning: a review of classification techniques. Emerg. Artif. Intell. Appl. Comput. Eng. 160, 3–24 (2007)

32. Quinlan, J.R.: Induction of decision trees. Mach. Learn. 1(1), 81–106 (1986)

33. Blum, A., Dwork, C., McSherry, F., Nissim, K.: Private mechanisms: the SULQ framework. In: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 128–138 (2005)

34. Friedman, A., Schuster, A.: Data mining with differential privacy. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 493–502 (2010)

35. Fletcher, S., Islam, M.Z.: Decision tree classification with differential privacy: a survey. ACM Comput. Surv. (CSUR) 52(4), 1–33 (2019)

36. Fletcher, S., Islam, M.Z.: A differentially private decision forest. AusDM 15, 99–108 (2015)

37. Fletcher, S., Islam, M.Z.: A differentially private random decision forest using reliable signal-to-noise ratios. In: Australasian Joint Conference on Artificial Intelligence, pp. 192–203. Springer, Berlin (2015)

38. Fletcher, S., Islam, M.Z.: Differentially private random decision forests using smooth sensitivity. Expert Syst. Appl. 78, 16–31 (2017)

39. Jagannathan, G., Pillaiappakamnatt, K., Wright, R.N.: A practical differentially private decision tree classifier. In: 2009 IEEE International Conference on Data Mining Workshops, pp. 114–121. IEEE (2009)

40. Patil, A., Singh, S.: Differential private random forest. In: 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 2623–2630. IEEE (2014)

41. Rana, S., Gupta, S.K., Venkatesh, S.: Differentially private random forest with high utility. In: 2015 IEEE International Conference on Data Mining, pp. 955–960. IEEE (2015)

42. Salzberg, S.L.: C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc. Mach Learn 16, 235–240 (1993)

43. Manton, K.G.: National long-term care survey: 1982, 1984, 1989, 1994, 1999, and 2004. Inter-university Consortium for Political and Social Research (2010)

44. Series, I.P.U.M.: Version 6.0. University of Minnesota, Minneapolis (2015)

45. Blake, C.L., Merz, C.J.: UCI Repository of Machine Learning Databases. University of California, Oakland (1998)