Sensitivity Analysis for Non-Interactive Differential Privacy: Bounds and Efficient Algorithms

Ali Inan, Mehmet Emre Gursoy, and Yucel Saygin

Abstract—Differential privacy (DP) has gained significant attention lately as the state of the art in privacy protection. It achieves privacy by adding noise to query answers. We study the problem of privately and accurately answering a set of statistical range queries in batch mode (i.e., under non-interactive DP). The noise magnitude in DP depends directly on the sensitivity of a query set, and calculating sensitivity was proven to be NP-hard. Therefore, efficiently bounding the sensitivity of a given query set is still an open research problem. In this work, we propose upper bounds on sensitivity that are tighter than those in previous work. We also propose a formulation to exactly calculate sensitivity for a set of COUNT queries. However, it is impractical to implement these bounds without sophisticated methods. We therefore introduce methods that build a graph model G based on a query set Q, such that implementing the aforementioned bounds can be achieved by solving two well-known clique problems on G. We make use of the literature in solving these clique problems to realize our bounds efficiently. Experimental results show that for query sets with a few hundred queries, it takes only a few seconds to obtain results.

Index Terms—Differential privacy, clique problems, statistical database security, SQL, range queries.

1 Introduction

Protecting individuals' private data against statistical disclosure attacks has been studied since the early 1980s [1]. On this subject, known as statistical database security, Dwork et al. have proven a very interesting conjecture: statistical database security cannot offer any absolute guarantees to individuals like *semantic security* does in cryptography [2]. In a semantically secure cryptosystem, a cipher-text does not reveal any information about the plain-text. The implications of Dwork's conjecture are very discouraging: regardless of the protection mechanism in place, every form of statistical interface to a private database brings together some risk of disclosure of private data. More fearsome is the fact that such disclosure might even harm persons whose record is not part of the database.

Differential privacy is a protection mechanism that was designed with this result in mind. Consider an individual, say Alice, who is trying to decide if she should place her record r into a statistical database $\mathcal{D}$. The two worlds resulting from this decision are as follows: (a) $\mathcal{D} \leftarrow \mathcal{D} \cup \{r\}$, (b) $\mathcal{D}' \leftarrow \mathcal{D} \cup \{r'\}$, where r' is the record of someone else (or some dummy record). Differential privacy encourages participation (world (a)) by minimizing the risks Alice will be taking.

This research was funded by The Scientific and Technological Research Council of Turkey (TUBITAK) under grant number 114E261. This manuscript extends previous work presented at SSDBM'16.

- *A. Inan is with the Department of Computer Engineering, Adana Science and Technology University, Turkey. E-mail: ainan@adanabtu.edu.tr*
- *M. E. Gursoy is with the College of Computing, Georgia Institute of Technology, Atlanta, GA, 30332. E-mail: memregursoy@gatech.edu*
- *Y. Saygin is with the Department of Computer Science and Engineering, Sabanci University, Turkey. E-mail: ysaygin@sabanciuniv.edu*

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TDSC.2017.2734664

Let $\mathcal{A}$ be a statistical data analysis task, e.g., a set of statistical queries. ε-Differential Privacy (ε-DP) [3] offers Alice exactly the following guarantee: the probability ratio that $\mathcal{D}$ and $\mathcal{D}'$ give the same results to $\mathcal{A}$ is bounded by e^{ε}. In the Laplace mechanism, this is achieved by adding noise to query responses. The noise magnitude depends on ε, and the L_1 sensitivity of the query set Q. This value, denoted $S_{L_1}(Q)$, is the largest effect of any single record (such as Alice's record r) on the responses to Q. $S_{L_1}(Q)$ is a function of the query set and does not depend on database $\mathcal{D}$.

Computing $S_{L_1}(Q)$ is not an easy task, since it requires studying the outcome of Q on all possible databases $\mathcal{D}$, $\mathcal{D}'$ differing in one record. Xiao and Tao prove in [4] that computing the sensitivity of a query set is NP-hard. This difficulty in part, and low utility of the results in general have led to the adoption of alternative approaches such as *smooth sensitivity* [5], and individual differential privacy [6] that measure sensitivity locally on the input database $\mathcal{D}$. Even though such definitions increase utility of the results, the original definition of DP [3] still offers the strongest privacy protection. An alternative to modifying the DP definition is to assume a safe, worst-case upper bound for $S_{L_1}(Q)$ that satisfies DP, but this as well yields high magnitudes of noise and destroys the utility of the private answers.

In this paper, we attempt to alleviate the difficulty of computing the sensitivity of a query set. We bound $S_{L_1}(Q)$ from above for statistical range queries, and present algorithms that efficiently realize these bounds. To the best of our knowledge, we find the tightest upper bound on $S_{L_1}(Q)$ available in the literature. We make our work open for public use through a web application that accepts SQL queries as inputs. Although there has been some work in calculating sensitivity for the likes of relational algebra [7], SQL is still by far the most popular query language in

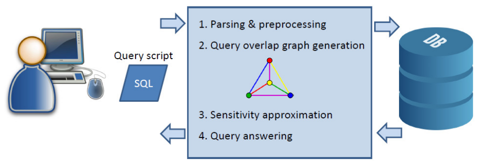
Query script
SQL
1. Parsing & preprocessing
2. Query overlap graph generation
3. Sensitivity approximation
4. Query answering
DB

Fig. 1: Work flow of the solution

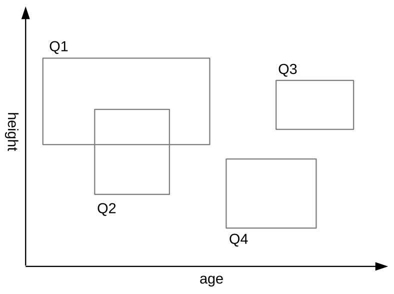

Fig. 2: Regions of queries in Q'

today's RDBMSs. Therefore, calculating $S_{L_1}(Q)$ for queries written in SQL is of great interest. Our solution is based on determining the ranges of statistical range queries, and using these ranges to convert Q to a graph. We then employ well-studied graph algorithms to approximate $S_{L_1}(Q)$.

1.1 Overview and contributions

The intended work flow of our solution is depicted in Fig. 1. We assume that an analyst (say, Bob) submits his query set Q to our differential privacy interface. Q will be parsed, and invalid queries (e.g., non-statistical) will be left out to build $Q' \subseteq Q$. The interface then approximates a value $S^{Q^0} \geq S_{L_1}(Q')$ and submits Q' to the RDBMS. Based on the privacy budget ε and the approximate sensitivity S^{Q^0}, the query answers will be perturbed with Laplace noise drawn from $\mathcal{L}(0, S^{Q^0}/\varepsilon)$ and returned to Bob. The interface currently works with statistical queries satisfying the grammar in Sec. 2.4, and databases with numeric, categorical and ordinal attributes.

S^{Q^0} is approximated using a graph $G(V, E)$ built from Q'. Suppose that Q' consists of the following queries on a 2-dimensional table T:

- $Q1$: `SELECT COUNT(*) FROM T WHERE Age BETWEEN 5 AND 30 AND Height BETWEEN 160 AND 190`
  $Q2$: `SELECT COUNT(*) FROM T WHERE Age BETWEEN 15 AND 25 AND Height BETWEEN 130 AND 170`
  $Q3$: `SELECT COUNT(*) FROM T WHERE Age BETWEEN 40 AND 50 AND Height BETWEEN 165 AND 185`
  $Q4$: `SELECT SUM(Age) FROM T WHERE Age BETWEEN 35 AND 45 AND Height BETWEEN 110 AND 155`

First, we determine the range of each query (i.e., query region) in Q'. We plot the regions of $Q1$-$Q4$ in Fig. 2. Using this plot, $G(V, E)$ is obtained as follows: We initially set $V = Q'$, i.e., each query is represented with a vertex in G. Two vertices are connected if their query regions intersect.

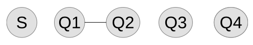

Fig. 3: Graph mapped from Q'

We add a *sink* vertex if need be (denoted with S in Fig. 3). The resulting graph in this case is shown in Fig. 3.

We show, theoretically, that it is possible to find two upper bounds on $S_{L_1}(Q')$ based solely on G. To the best of our knowledge, one of our bounds partially matches the bound in [4] and the other bound is tighter than those in previous works. We show that computing these bounds rely on solving two well-known clique problems: the maximum clique problem (MCP) and maximal clique enumeration problem (MCEP) on G. Even though these clique problems are NP-hard, they are heavily studied in the computer science literature, and there exist efficient algorithms that provide exact solutions. One of the primary strengths of our approach (i.e., modelling a query set as a graph) is the exploitation of these works.

The contributions of this work can be listed as follows:

- We provide theoretical upper bounds on the sensitivity of a set of statistical range queries (see Sec. 3). We elaborate on why these bounds are difficult to implement efficiently in a straightforward manner.
- To implement our bounds, we propose methods to map a given set of queries into a graph model. Our mapping requires no additional knowledge apart from the queries themselves and the domain of numerical attributes (e.g., age, height).
- We theoretically prove that our sensitivity bounds depend on solving two clique problems (MCP and MCEP) on our graph model. We utilize state-of-the-art libraries for solving the clique problems and experimentally show that our solutions can be implemented efficiently and easily.
- We provide a proof-of-concept implementation for a restricted but very expressive subset of standard SQL, in which graph generation and sensitivity calculation can be done automatically[1]. We expect the integration of this implementation into commercial RDBMSs to be straightforward, so that analysts can work with the familiar SQL interface.
- We acknowledge the limitations of our approach by showing that computing the *exact* sensitivity of a query set cannot be achieved using our graph model.

The rest of this paper is organized as follows: In Sec. 2, we describe our assumptions regarding the data and query models, and give a brief introduction to differential privacy and clique problems. In Sec. 3, we present our sensitivity bounds. In Sec. 4, we explain how a query set Q can be modelled as a graph G. We then prove some useful properties of the graph, including the theorems necessary to convert our sensitivity bounds into clique problems. In Sec. 5, we describe and discuss the algorithms used for bounding sensitivity via G. Implementation details and experimental results on the efficiency of our solution are given in Sec. 6. We review the related work in Sec. 7 and conclude in Sec. 8.

2 PRELIMINARIES

2.1 Assumptions on the database schema

We consider databases $\mathcal{D}$ consisting of a single table with d attributes, denoted $A_1, A_2, ..., A_d$. We use the term *dimension*

1. Please see http://sky.sabanciuniv.edu:8000/

to refer to these attributes, e.g., a database is d-dimensional if and only if it contains d attributes. The domain of attribute A_i is denoted with $\Omega(A_i)$. The domain of the whole database is denoted with $\Omega(\mathcal{D})$.

We make the following (widely accepted) assumptions on the schema of $\mathcal{D}$:

- For each attribute A_i, the domain $\Omega(A_i)$ is finite. Finite domains allow bounding the effect of a single record on the output of domain-specific aggregate functions, such as SUM.
- Attributes are either numeric, categorical or ordinal. Some attribute types (e.g., binary objects, dates) can be easily transformed into numeric values. Other attribute types (e.g., strings) cannot be supported, due to the difficulty in converting their domain into a finite, well-defined set of values.
- Domains of numeric attributes are normalized to the range $[0, 1)$. This requirement removes any domain dependence in the sensitivity analysis. It can be achieved trivially when $\Omega(A_i)$ is finite, and $\min(\Omega(A_i))$ and $\max(\Omega(A_i))$ are known in advance.

2.2 Differential Privacy

Differential privacy aims to ensure that the result of a data analysis is not overly dependent on any single data record. To achieve this, it conjectures that there should be a strong probability that a differentially private query answering algorithm produces the same results even if one record in the database was changed. The definitions below formalize this notion.

Definition 1 (Neighboring databases). Two databases $\mathcal{D}$, $\mathcal{D}'$ are called neighboring databases, if they have the same schema and cardinality, and differ in only one record.

Definition 2 (ε-Differential privacy). A randomized algorithm $\mathcal{A}$ is ε-differentially private (ε-DP) if for all neighboring databases $\mathcal{D}$, $\mathcal{D}'$ and for all possible outcomes of the algorithm $S \subseteq Range(\mathcal{A})$,

$$Pr[\mathcal{A}(\mathcal{D}) \in S] \leq e^{\varepsilon} \times Pr[\mathcal{A}(\mathcal{D}') \in S]$$

where the probabilities are over the randomness of $\mathcal{A}$.

A variant of the above definition is where a neighboring database $\mathcal{D}'$ is obtained by adding one record r to database $\mathcal{D}$, or by removing a record r from $\mathcal{D}$. This variant was later named *unbounded differential privacy (UDP)* [8], whereas the variant we use was named *bounded differential privacy*. The remainder of this paper assumes bounded differential privacy as it is used in the initial definition of ε-DP [3]. We explicitly note those cases where we refer to UDP.

In ε-DP, the user poses a set Q of statistical queries to a database, which are then answered by adding independent random noise to the true output of each query. The noise is calibrated according to the L_1 *sensitivity* of the query set.

Definition 3 ($S_{L_1}(Q)$: L_1 Sensitivity of Q). Let $q(\mathcal{D})$ denote the output of query q on database $\mathcal{D}$. Given a set of queries Q, the sensitivity of Q, denoted $S_{L_1}(Q)$, is:

$$S_{L_1}(Q) = \max_{\mathcal{D},\mathcal{D}^0}(\sum_{q \in Q} |q(\mathcal{D}) - q(\mathcal{D}')|)$$

where $\mathcal{D}$, $\mathcal{D}'$ are any two neighboring databases.

A popular ε-DP algorithm to answer numeric queries is the Laplace mechanism [3]. In the Laplace mechanism, random noise is sampled from the Laplace distribution. The scale parameter of the distribution is determined by the privacy budget ε and $S_{L_1}(Q)$, as defined below.

Definition 4 (Laplace mechanism). Let $Lap(0, \lambda)$ denote the Laplace distribution with mean 0 and scale parameter λ. For a set Q of real-valued queries, let $Q(\mathcal{D})$ denote the output vector (i.e., transcript) of Q on $\mathcal{D}$. An algorithm $\mathcal{A}$ that outputs $\mathcal{A}(Q, \mathcal{D}) = Q(\mathcal{D}) + N$ where N is a random noise vector with each entry sampled from $Lap(0, \lambda)$ satisfies ε-DP if $\lambda \geq S_{L_1}(Q)/\varepsilon$.

We refer to λ as the noise magnitude. Based on this definition, from a privacy point of view, it is fine to overestimate $S_{L_1}(Q)$. This would only cause the noise magnitude to be higher than it actually could be, but would nevertheless satisfy ε-DP. However, this is not desirable from a utility point of view, because query outputs would be more noisy than theoretically necessary.

For example, let Bob have $|Q| = 100$ count queries. Being a naive user, Bob decides to play safe and assume that his query set has sensitivity 100, whereas $S_{L_1}(Q)$ is actually 30. Bob sets $\lambda = 100/\varepsilon$ and ends up getting answers that have excess noise, which deteriorates the quality of his results. If he had known that $S_{L_1}(Q) = 30$, he could have set $\lambda = 30/\varepsilon$ and obtained less noisy, and consequently more accurate results using the same ε as before.

2.3 Clique Problems

Since our work is based on modelling query sets as graphs, in this section we give a brief introduction to graph terminology and the clique problems.

Let $G(V, E)$ be an undirected graph with vertex set V and edge set $E \subseteq V \times V$. A *clique* C of G is a subset of V such that every two vertices in C are adjacent, i.e., $\forall u, v \in C, (u, v) \in E$. A *maximal clique* is a clique to which no more vertices can be added. In other words, a maximal clique is not contained by any other clique. A clique is a *maximum clique* if its cardinality is the largest among all the cliques of the graph. A maximum clique is also maximal. A graph may contain multiple maximum cliques.

Definition 5 (Maximum clique problem). Given a graph $G(V, E)$, the maximum clique problem is to find a clique C of G that has the highest cardinality. We denote the cardinality/size of C, often called the clique number of G, with $MCS(G)$.

For example, in Fig. 5, {Q3, Q4} is a maximal clique, but it is not maximum. Clique {Q1, Q2} is neither maximal, nor maximum. {Q1, Q2, Q3} is a maximum clique, and there is no other maximum clique. For the graph G, $MCS(G) = 3$.

The maximum clique problem (MCP) has a wide range of applications, and is among the most heavily studied combinatorial problems. Even though the MCP is NP-complete [9], due to its practical relevance, there has been significant effort for finding efficient solutions. We refer the interested reader to [10] for a recent survey on algorithms for the MCP.

Another interesting clique problem is *maximal clique enumeration*. The goal is to list all maximal cliques in a graph.

Definition 6 (Maximal clique enumeration problem). Given a graph $G(V, E)$, the maximal clique enumeration problem is the process of listing all maximal cliques of G. We denote the set of all maximal cliques of G with $MCE(G)$.

For example, in Fig. 5, $MCE(G)$ consists of {Q1, Q2, Q3}, {Q3, Q4}, {S} and {Q5}. Notice that we treat single vertices also as cliques, of size 1.

The maximum clique enumeration problem (MCEP) is relatively less heavily studied compared to the MCP, due to the inherent difficulty in not only finding *all* maximal cliques but also listing/storing them. [11] shows that an n-vertex graph can have as high as $3^{n/3}$ maximal cliques. Although there are algorithms for the MCEP that have $O(3^{n/3})$ time complexity (thus matching the optimal time-bound) [12][13], their applications have mostly focused on sparse graphs, where the maximal cliques themselves can fit into memory [14].

2.4 Statistical Range Queries

Recall that we treat the schema of a database $\mathcal{D}$ as d-dimensional space: $\Omega(\mathcal{D}) = \Omega(A_1)\times\Omega(A_2)\times...\times\Omega(A_d)$. Our work is concerned with statistical range queries that are essentially d-dimensional hyper-rectangles in $\Omega(\mathcal{D})$. Statistical range queries can be written and evaluated in an arbitrary language, i.e., our methods are not platform-dependent. Yet, in this work we use the Structured Query Language (SQL) to formally introduce them, due to SQL's widespread use and well-defined grammar.

We say that a statistical range query is in the form:

```
SELECT AGG
FROM T
WHERE pred(A1) AND ... AND pred(Ad)
```

where AGG is any valid SQL aggregate function but AVERAGE(A_i), which we suggest be queried explicitly through a SUM(A_i) followed by a COUNT(*). pred(A_i) is a predicate on attribute A_i. Allowed predicates are below.

- A_i op x, where $x \in \Omega(A_i)$ and op $\in \{=, >, <, \leq, \geq\}$,
- A_i BETWEEN (x, y), where $x, y \in \Omega(A_i)$,
- pred(A_i) is omitted, i.e., no constraints on the i^{th} attribute.

Notice that the predicates are chosen such that the condition on A_i expresses an interval[2] in $\Omega(A_i)$. Since disjunctions (i.e., OR) are disallowed in the selection condition, any query in the above grammar has a query region that is a single hyper-rectangle[3] in $\Omega(D)$.

We would like to stress that some of the restrictions imposed by the above grammar are placed for purely practical reasons. Graph modelling of a query set involves intersecting the regions of queries (see Sec. 4.1). If a query region is a d-dimensional hyper-rectangle, then checking for intersecting regions becomes a trivial task. Theoretically, even if query regions are non-rectangular, our solutions would still be applicable. Consider Fig. 2 and assume that the *age* and *height* axes respectively denote x and y coordinates of the Cartesian plane. Let each query q_i be formed with a circular range as "points within r_i Euclidean distance from center point p_i". Modifying Alg. 1 to decide whether the regions of some q_i and q_j intersect would be sufficient. Alg. 2 and all of our bounds would apply directly.

Statistical range queries are the workhorse of many popular data analysis tasks including histograms, classification (e.g., Naive Bayes classifiers) and association rule mining. Accurately and privately answering COUNT queries with rectangular ranges alone has been a primary area of research over the last decade [15][4].

We provide an open, testable implementation of our work (which will be detailed in Sec. 6). This implementation requires that queries be written in SQL. Another advantage of using SQL is that we can employ an SQL parser to check if given queries comply with the requirements above. Any query that does not fit into this grammar is identified by the parser and eliminated from the sensitivity analysis. This also applies to non-statistical queries that try to retrieve raw data from the database.

3 Sensitivity Analysis

Sec. 2.1 lists some basic assumptions on the schema of the target database D to be protected according to ϵ-DP. Sec. 2.4 lists the properties of statistical queries that can be handled by our solution. In this section, we theoretically analyze the sensitivity $S_{L_1}(Q)$ of a supported Q over a database D whose schema is supported.

Differential privacy defines the sensitivity of a query set over all neighboring databases D and D', where each differ from the other in only one record (please see Def. 2 and Def. 3). Let T be the set of records common to D and D' and, r and r' denote the records that are different. Specifically, $T = D \setminus D'$, $r = D - D'$, and $r' = D' - D$.

The analyses will be carried out over arbitrary records r and r' of the database schema $\Omega(D)$. Our primary tool for the analysis will be the *popularity set* of a record, introduced in [4] and formally defined below. In the definition, we refer to the range (i.e., query region) of a query q as $range_q$.

Definition 7 (Popularity set of a record r: $popSet(r)$ [4]). Given a set Q of queries and a record r, the popularity set of r, denoted $popSet(r)$, is the set of all queries q in Q, whose range contains r. Formally:

$$popSet(r) = \{q \in Q : r \in range_q\}$$

Informally, $popSet(r)$ is the subset of queries of Q each of whose answer depends on r.

We first make an important observation: due to our assumptions on $\Omega(D)$ and Q, the effect of a single record change (i.e., $r \to r'$) on a query q can be bounded easily.

Theorem 1. [16] For any query q and any neighboring databases D, D', $|q(D) - q(D')| \leq 1$.

The proof is available in [16] and relies heavily on the domain normalization assumption of Sec. 2.1. We would like to stress that, in cases where $\Omega(A_i)$ is finite, this assumption has no effect on how query results are perturbed. Consider the following very simple example: $\Omega(Age) = [0, 110)$. The sensitivity of a query q asking for MIN(Age) over records satisfying some predicate P will be 110. $q(D)$ will be perturbed with noise drawn from $L(0, 110/\epsilon)$. If Age is normalized into Age', sensitivity becomes 1 and the (normalized) true response ($= q(D)/110$ after normalization) will be perturbed with $L(0, 1/\epsilon)$. Mathematically, these two cases are equivalent because noise is additive. Yet, the analysis for the latter case is much more simpler to follow.

When $\Omega(A_i)$ is infinite, our domain normalization assumption cannot hold. However, this situation is not a deficiency of our solution. ϵ-DP does not support querying of such attributes. Suppose $\Omega(Att)$ is infinite. Sensitivity of the query SELECT MIN(Att) ... is infinite regardless of the query predicate. Added noise will be picked from $L(0, \infty/\epsilon)$, in

2. A point p in $\Omega(A_i)$ is the interval $[p, p]$
3. We consider planes and points in d-dimensions to also be hyper-rectangles since this does not affect the correctness of our analyses.

which case, the perturbed query response will be theoretically completely random.

In Sec. 3.1, we use popularity sets of records to provide the tightest known bound on $S_{L_1}(Q)$. Then, in Sec. 3.2, we prove an exact solution for the case that Q contains only `COUNT` queries.

3.1 Bounding $S_{L_1}(Q)$ based on $popSet(r)$

Theorem 2. For any pair of neighboring databases D, D', let $r = D - D'$ and $r' = D' - D$. Then, for a query set Q, $S_{L_1}(Q) \le \max_{r,r'} |popSet(r) \cup popSet(r')|$, where $r, r' \in \Delta(D)$.

Proof: Let $popSet(r)$ and $popSet(r')$ denote the popularity sets of r and r' respectively. We partition the queries q in Q into 4 mutually exclusive and collectively exhaustive sets based on whether $q \in popSet(r)$ and $q \in popSet(r')$.

$pop_{r\neg r'} = popSet(r) - popSet(r')$.
$pop_{\neg r r'} = popSet(r') - popSet(r)$.
$pop_{r r'} = popSet(r) \cap popSet(r')$.
$pop_{\neg r\neg r'} = Q - (popSet(r) \cup popSet(r'))$.

$S_{L_1}(Q)$ is computed over all neighboring databases D and D'. We need to maximize the total L_1-distance to individual query responses. For a single query, we denote this distance with δ. That is, we use $\delta = |q(D) - q(D')|$ and start with the definition of $S_{L_1}(Q)$ given in Def. 3:

$$S_{L_1}(Q) = \max_{D,D'} \left(\sum_{Q} \delta \right) \tag{1}$$

$$= \max_{D,D'} \left(\sum_{pop_{r\neg r'}} \delta + \sum_{pop_{\neg r r'}} \delta + \sum_{pop_{r r'}} \delta + \sum_{pop_{\neg r\neg r'}} \delta \right) \tag{2}$$

$$= \max_{D,D'} \left(\sum_{pop_{r\neg r'}} \delta + \sum_{pop_{\neg r r'}} \delta + \sum_{pop_{r r'}} \delta + 0 \right) \tag{3}$$

$$\le \max_{r,r' \in \Delta(D)} \left(\sum_{pop_{r\neg r'}} 1 + \sum_{pop_{\neg r r'}} 1 + \sum_{pop_{r r'}} 1 \right) \tag{4}$$

$$\le \max_{r,r' \in \Delta(D)} \left(\sum_{popSet(r) \cup popSet(r')} 1 \right) \tag{5}$$

$$\le \max_{r,r' \in \Delta(D)} |popSet(r) \cup popSet(r')| \tag{6}$$

Eq. 2 breaks the summation in Eq. 1 down by expressing it in terms of popularity sets of r and r'. In Eq. 2, we observe that if a query is in $pop_{\neg r\neg r'}$, i.e., its answer is affected neither by r nor r', then $\delta = 0$. We therefore obtain Eq. 3. To move from Eq. 3 to Eq. 4, we make use of Th. 1: for all queries affected by a single record change $r \rightarrow r'$, $\delta \le 1$. Since possible effects of $T = D \cap D'$ are contained in this transition, we also switch to a maximization over records r and r'. The rest of the proof is straightforward, as Eq. 5 joins the 3 sums into a single sum and Eq. 6 provides a closed form formula to this sum. □

3.2 The case of $COUNT$ queries

When Q contains only counting queries, a slightly modified version of the above analysis gives an exact solutions to the sensitivity of Q. We observe that when $q \in Q$ is a count query, any record r displaces $q(D)$ by either 0 or 1. There is no other alternative, which simplifies the analysis.

Theorem 3. For a query set Q consisting only of `COUNT` queries, $S_{L_1}(Q) = \max_{r,r'} |popSet(r) - popSet(r')| + |popSet(r') - popSet(r)|$, where $r, r' \in \Delta(D)$.

Proof: Similar to the proof above, we have:

$$S_{L_1}(Q) = \max_{D,D'} \left(\sum_{pop_{r\neg r'}} \delta + \sum_{pop_{\neg r r'}} \delta + \sum_{pop_{r r'}} \delta + \sum_{pop_{\neg r\neg r'}} \delta \right) \tag{7}$$

$$= \max_{D,D'} \left(\sum_{pop_{r\neg r'}} 1 + \sum_{pop_{\neg r r'}} 1 + \sum_{pop_{r r'}} 0 + \sum_{pop_{\neg r\neg r'}} 0 \right) \tag{8}$$

$$= \max_{r,r' \in \Delta(D)} |popSet(r) - popSet(r')| + |popSet(r') - popSet(r)| \tag{9}$$

Q contains only `COUNT` queries. For certain, for queries in $pop_{r\neg r'}$ and $pop_{\neg r r'}$, we have $\delta = 1$. The response to queries in $pop_{r r'}$ or $pop_{\neg r\neg r'}$ will be the same over D and D', so we have $\delta = 0$. The maximization is over any possible pairs of records from $\Delta(D)$. □

3.3 Discussion on $popSet$s

The analyses above yield upper bounds on the sensitivity of a query set, and these bounds are tighter than all existing bounds available in the literature [4]. Unfortunately, unless $\Delta(D)$ is discrete and very small, the bounds above have only theoretical value. They are hardly of any practical significance because the maximization is over all pairs of records (r, r') in the database domain, which is impractical to exhaustively generate. Furthermore, if one or more attributes in D is continuous rather than discrete, it becomes impossible to generate all possible records in $\Delta(D)$.

A simple observation greatly reduces the complexity of the maximization problems in Eq. 5 and Eq. 9. For both problems, the maximization domain appears to be $\Delta(D)$. However, for any record r'' such that $popSet(r'') \equiv popSet(r)$, the outcomes of the objective functions will be the same. This implies that, the maximization problems can essentially be expressed over pairs of popularity sets of the query set Q. If we had a method of listing all possible popularity sets of Q, generating all pairs of records would not be necessary.

In Sec. 4, we try to achieve this by extracting a graph model of the input query set Q. Sec. 5 shows that some of the above bounds based on $popSet$s can be realized with the extracted graph. Furthermore, these realizations rely on solutions to very well known and heavily studied problems of graph theory. Sec. 5.5 discusses the limitations of this approach, i.e., why only some bounds can be supported by the graph, but not all.

4 GRAPH MODELLING OF A QUERY SET

We start with some notation on a single query q, and a query set Q_s. Throughout this section, we assume that both q and elements of Q_s are statistical range queries that fit the grammar given in Sec. 2.4. The notation that will be used this section onwards is summarized in Tab. 1.

Let q be a query that contains a selection condition expressed in the `WHERE` clause, denoted by $q.where$. The predicate on a specific attribute A_i can be fetched through an index on the attributes, as in $q.where[A_i]$, which is an interval (i.e., a range) on $\Delta(A_i)$. This interval is denoted by $range_q^{A_i}$. In d-dimensional space, the range of query q becomes a d-dimensional hyper-rectangle, which we denote by $range_q$.

Based on this notation, we make the following definition of the range-intersection of a set of queries.

Definition 8 (Range-intersection of a set of queries). For a query set Q_s such that $|Q_s| > 1$, the range-intersection is denoted with $range_{Q_s}$ and represents a range that is contained by the ranges of all elements of Q_s. That is:

$$range_{Q_s} = \bigcap_{q \in Q_s} range_q$$

q	Query
$q.where$	WHERE condition of q
$q.where[A_i]$	Condition of q on attribute A_i
Q or Q_s	Set of queries
$range_q$	Range of q
$range_q^{A_i}$	Range of q on attribute A_i
$range_{Q_s}$	Range-intersection of queries in Q_s
$S_{L_1}(Q_s)$	L_1 sensitivity of Q_s
$G(V, E)$ or G	Graph
$MCS(G)$	Maximum clique size of G
$MCE(G)$	Maximal clique enumeration on G

TABLE 1: Our notation

Essentially, the range-intersection is the common intersection of all queries in Q_s. For example, in Fig. 4, if Q_s = fQ1; Q2; Q3g, then $range_{Q_s}$ is the area denoted 3a. If Q_s = fQ1; Q2; Q4g, then $range_{Q_s}$ is empty. If Q_s = fQ3; Q4g, then $range_{Q_s}$ is equal to $range_{Q4}$.

4.1 Graph generation

In Alg. 2, we outline a mapping algorithm that generates an undirected graph $G(V; E)$ from a set Q of queries. The graph contains one vertex for each query in Q. The edge-set E of the graph G is constructed based on a function given in Alg. 1 that determines whether the query regions of a pair of queries $(p; q)$ intersect or not.

Algorithm 1 Comparing regions of queries p and q

```
1: function INTERSECTS(Query $p$, Query $q$)
2:     for Each att. $A_i$ listed in both $p.where$
        and $q.where$ do
3:         $range_p^{A_i} \leftarrow p.where[A_i]$
4:         $range_q^{A_i} \leftarrow q.where[A_i]$
5:         if $range_p^{A_i} \cap range_q^{A_i} = \emptyset$ then
6:             return false
7:     return true
```

Alg. 1 operates on attributes A_i that are referenced in the `where` clause of both queries. In other words, both queries contain a predicate on attribute A_i, i.e., `pred(`A_i`)`. For each such attribute, the corresponding ranges are retrieved in steps 3 and 4. The regions of queries p and q intersect if and only if they intersect on every attribute A_i of the table. If $p{:}where$ conditions on an attribute A_i but $q{:}where$ does not, we conclude that $q{:}where[A_i]$ = (1 ; 1) and the intersection on dimension A_i is non-empty trivially.

The mapping algorithm that generates the actual graph is given in Alg. 2. Queries are inserted into V in steps 3-4. Edges are inserted into E in steps 6-9. For each possible pair of queries $(p; q)$, a call to Alg. 1 is made. If the query regions intersect, then in G, vertices p and q will be connected.

Before explaining further steps of Alg. 2, let us give some intuition about what information G tries to capture. A query-vertex q 2 Q added at step 4 of the algorithm represents all records r in (D) such that $popSet(r)$ = fqg. Similarly, a clique C of G (which itself can be seen as a subset of Q) represents all records r with $popSet(r) = C$.

When Alg. 2 reaches step 10, only records relevant to some q 2 Q will be encoded in G. In steps 10-13, we first check whether Q covers (D). If not, in step 16, we insert a special sink vertex S to represent records with empty $popSet$s. Adding S is convenient because we can easily argue for the completeness/correctness of our analyses when we have a sink vertex S. The sink vertex S has no edges. By construction, the subdomain S represents does not overlap with any query range from Q.

Algorithm 2 Mapping Q to $G(V, E)$

```
1: function GEN-GRAPH(Query set $Q$)
2:     $V \leftarrow \emptyset$
3:     for Each query $q \in Q$ do
4:         $V \leftarrow V \cup \{q\}$
5:     $E \leftarrow \emptyset$
6:     for Each query $p \in Q$ do
7:         for Each query $q \in Q, p \neq q$ do
8:             if INTERSECTS($p, q$) then
9:                 $E \leftarrow E \cup \{(p, q)\}$
10:    $covers_Q \leftarrow$ true
11:    for Each attribute $A_i \in Q$ do
12:        if $\cup_{q \in Q} range_q^{A_i} \neq \Omega(A_i)$ then
13:            $covers_Q \leftarrow$ false
14:    if $covers_Q =$ false then
15:        Create a sink vertex $S$
16:        $V \leftarrow V \cup \{S\}$
17:    return $G(V, E)$
```

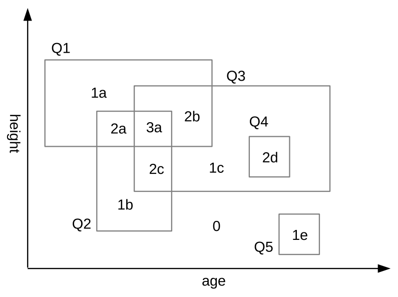

Fig. 4: Regions of queries in Q

At this point we study the examples in Fig. 4 and Fig. 5. Suppose our query set is Q = fQ1; :::; Q5g with the ranges plotted in Fig. 4. $range_{Q1}$ and $range_{Q2}$ intersect in both dimensions (the intersection is the union of areas denoted 2a and 3a) and therefore there is an edge between $Q1$ and $Q2$ in Fig. 5. On the other hand, if we study $Q3$ and $Q5$, we observe that $range_{Q3}^{age}$ and $range_{Q5}^{age}$ intersect, but $range_{Q3}^{height}$ and $range_{Q5}^{height}$ do not. Hence, $range_{Q3}$ \ $range_{Q5}$ = ;. Consequently, in Fig. 5, there is no edge between $Q3$ and $Q5$. If we study $Q3$ and $Q4$, we observe that $range_{Q4}$ $range_{Q3}$, hence they have a common intersection, $range_{Q4}$. Therefore in Fig. 5 $Q3$ and $Q4$ are connected to one another. Lastly, we see that our query set Q does not cover the whole data space (D) (see the area denoted 0). Thus we include a sink vertex S in Fig. 5. The sink vertex is not connected to any of the remaining vertices.

Complexity of Alg. 1 is $O(d)$, where d is the dimensionality of the table. Alg. 2 calls this function for each pair of queries. Consequently, the overall complexity of generating G from Q is $O(d$ jQj$^2)$.

4.2 Some useful properties of the graph

The graph generated according to Alg. 2 for a query set Q has some properties that will be useful for bounding the sensitivity

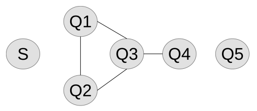

Fig. 5: Graph mapped from Q

of Q in Sec. 5. In this section, we present these properties. Proofs of Th. 4 and Th. 5 are available in [16].

Theorem 4. For vertex set $Q_s \subseteq V$ such that $|Q_s| > 1$, if Q_s is a clique of G, then the range-intersection of the queries represented by Q_s is non-empty. Formally:

$$Q_s \times Q_s \subseteq E \implies range_{Q_s} \neq \emptyset$$

Theorem 5. For a query set Q_s such that $|Q_s| > 1$, if the range-intersection of the queries is non-empty, then Q_s represents a clique of graph G. Formally:

$$range_{Q_s} \neq \emptyset \implies Q_s \times Q_s \subseteq E$$

Together, Th. 4 and Th. 5 indicate the equivalence of the two problems: finding a clique of the graph G built according to Alg. 2 and finding a subset of queries in an input query set whose range-intersection is non-empty.

We go back to Fig. 4 and 5 to illustrate this with examples. We observe that $\{Q1, Q2, Q3\}$ and $\{Q3, Q4\}$ are cliques in the graph, and their range-intersections are 3a and 2d respectively (i.e., they have non-empty range-intersections). Subsets of these cliques are also cliques, e.g., $\{Q1, Q2\}$ constitute a clique, and their range-intersection is the area $(2a \cup 3a)$. Furthermore, $\{Q1, Q3\}$ is a clique with a range-intersection denoted $(3a \cup 2b)$ in Fig. 4. Continuing in this fashion, one can see that all cliques have a non-empty common intersection.

Finally, we establish a relation between cliques of G and the popularity set of a record r. According to Th. 6, every distinct $popSet$ is a clique of the mapped graph G and there is a one-to-one mapping (i.e., an injection) from $popSet$s to cliques.

Theorem 6. Let $r \in \mathcal{D}(D)$ be a record. For a given query set Q, $popSet(r)$ can be mapped to one and only one clique of G extracted from Q according to Alg. 2.

Proof: We consider two cases: $popSet(r) = \emptyset$ and $popSet(r) \neq \emptyset$. If $popSet(r) = \emptyset$, then $\forall q \in Q$, $r \notin range_q$. Recall that Alg. 2 creates a special sink vertex S for all such records r. In this case the theorem is satisfied since G represents r (and $popSet(r)$) via S.

If $popSet(r) \neq \emptyset$, then $\forall q \in popSet(r)$, $r \in range_q$ and consequently, $r \in range_{popSet(r)}$. That is, r is in the range-intersection of all queries in $popSet(r)$. Since such an r exists, $range_{popSet(r)} \neq \emptyset$ and based on Th. 5, $popSet(r)$ should be a clique of G.

For both cases, given a record r, $popSet(r)$ with respect to Q is unique and is represented by one and only one clique of the mapped graph G. □

The proof above will be critical in Sec. 5. As discussed in Sec. 3.3, realizing the bounds on $popSet$s is impractical because these require generating all possible records $r \in \mathcal{D}(D)$. If, however, the bounds can be related to certain properties of the mapped graph G, this impracticality can be overcome.

Let us illustrate the implications of Th. 6. In Fig. 4, if we place r anywhere in 3a, its $popSet(r) = \{Q1, Q2, Q3\}$ and this is a clique in Fig. 5. There is no other clique in Fig. 5 that represents the area labeled 3a (hence, $popSet(r)$ is mapped to only one clique in the graph). If we place r in 2a, $popSet(r) = \{Q1, Q2\}$ and this is a clique. If we place r in 1b, $popSet(r) = \{Q2\}$, and we consider this a single-vertex clique. If we place r in 0, i.e., outside the range of every query, its $popSet(r) = \{S\}$, and this is a single-vertex clique.

5 Bounding sensitivity using the graph model

The graph mapping G of a query set Q allows bounding the sensitivity of Q in multiple ways. In this section, we introduce these bounds and accompanying algorithms.

5.1 Bounding $S_{L_1}(Q)$ with $|Q|$

Recall that in Th. 1 we bounded the change in a query's output due to a single record change. A straightforward application of this theorem gives a crude upper bound on $S_{L_1}(Q)$.

Theorem 7. [16] For any query set Q, under the assumptions of Sec. 2.4, $S_{L_1}(Q) \leq |Q|$.

The proof of Th. 7 is available in [16]. The bound established by this theorem is global - in the following sense: as long as domains of attributes are normalized into range $[0, 1)$, regardless of whether queries have rectangular ranges or not, this bound can be applied[4].

5.2 Bounding $S_{L_1}(Q)$ with $MCS(G)$

The bound given in Sec. 5.1 requires almost no computation at all but is very crude. Our second bound limits $S_{L_1}(Q)$ with the size of the largest clique in G - known as $MCS(G)$ - the maximum clique size of G.

Theorem 8. For any query set Q, $S_{L_1}(Q) \leq 2 \times MCS(G)$, where G is the graph generated according to Alg. 2 and $MCS(G)$ represents the size of the maximum clique of G.

Proof:

$$S_{L_1}(Q) \leq \max_{r, r' \in \mathcal{D}(D)} |popSet(r) \cup popSet(r')| \tag{10}$$

$$\leq \max_{r \in \mathcal{D}(D)} |popSet(r)| + \max_{r' \in \mathcal{D}(D)} |popSet(r')| \tag{11}$$

$$\leq \max_{\text{clique } C \text{ of } G} |C| + \max_{\text{clique } C \text{ of } G} |C| \tag{12}$$

$$\leq 2 \times MCS(G) \tag{13}$$

Eq. 10 is the same as Eq. 5 of Th. 2. Eq. 11 loosens the bound by the size of $|popSet(r) \cap popSet(r')|$, which is guaranteed to be non-negative. Since every $popSet$ is a clique (Th. 6), Eq. 12 changes the maximization domain from $popSet$s to cliques of G. By definition, the size of the largest clique of G is $MCS(G)$. □

Alg. 3 outlines our first bound over $S_{L_1}(Q)$ based on the graph modelling of Q.

Algorithm 3 Bounding $S_{L_1}(Q)$ with $MCS(G)$ and $|Q|$

```
1: function SENS-MCS(Query set $Q$)
2:     $G \leftarrow$GEN-GRAPH($Q$)
3:     $MCS \leftarrow MCS(G)$
4:     return $\min(|Q|, 2 \times MCS)$
```

Proof of correctness: The algorithm follows directly from Th. 7 and Th. 8. Let us look at an interesting case, where $Q = \emptyset$. We have $MCS(G) = 1$ due to the sink vertex S. However, $|Q| = 0$ and Alg. 3 correctly returns 0 at step 4. Using the $2 \times MCS(G)$ bound by itself would (again, correctly) overestimate $S_{L_1}(Q)$ as 2. Notice that $Q = \emptyset$ is the only case where the return value of Alg. 3 depends entirely on the sink vertex S.

Alg. 3 is very simple but expresses the main advantages of our approach. Graph G encloses all necessary information needed to realize this sensitivity bound. In addition, by separating the graph generation and $MCS(G)$ finding steps, we allow the plethora of work on computing $MCS(G)$ to be directly applicable to approximate $S_{L_1}(Q)$.

Next, we discuss the intuition behind our sensitivity bound. We present a change in one record (per the definition of neighboring databases) as $r \to r'$. This can be thought of as removing r from the database and adding r' instead. The effect of this operation is maximized when both r and r' affect a maximum number of queries. That is, if r and r' are in the

4. Disjunctive predicates in the selection condition (i.e., using OR) are still disallowed.

range-intersection of a large number of queries, then all those queries will be affected by $r \to r'$. Since cliques are equivalent to range-intersections, a maximum clique yields an area of range-intersection that affects the maximum number of queries.

For the example in Fig. 4 and Fig. 5, the maximum clique is $C_1 = \{Q1, Q2, Q3\}$. Therefore, we place $r \in range_{C_1}$ (i.e., r is in range-intersection of $\{Q1, Q2, Q3\}$, denoted 3a). Removal of r from D will affect 3 queries, which is the maximum impact it can do. The $2 \cdot |MCS(G)|$ bound, however, (implicitly) assumes that there is a second maximum clique C_2 in which r' can be placed. If there were such a C_2, then since it is maximum $|C_1| = |C_2|$ and the impact of $r \to r'$ would be twice the impact of r, hence 6. However, in this example there is no such C_2, hence this bound is an overestimation. As discussed earlier, an overestimation is not a problem from a privacy point of view, but it is undesirable from a utility point of view. Indeed, the next largest clique in Fig. 5 is $\{Q3, Q4\}$. A sensitivity of 6 cannot be obtained from this example - it can clearly be seen that Fig. 4 contains 5 queries, thus $S_{L_1}(Q) \le 5$.

We note that (r, r') is symmetric, i.e., if we interchanged r and r' we would have obtained the same result. We also note that a higher change in the output cannot be obtained by placing (r, r') in areas with less common intersections, e.g., if we place r in 1a and r' in 1e. In this case, the effect of $r \to r'$ would be 2. But, the definition of L_1 sensitivity is concerned with the *maximum* possible change, hence this is not useful.

5.3 Bounding $S_{L_1}(Q)$ with $MCE(G)$

The previous bound implicitly assumes that there are two disjoint maximum cliques of G (one that covers r of D and another that covers r' of D'). This may not be the case for various reasons: (a) the second-largest clique of G may be much smaller, and (b) the intersection of these largest maximal cliques may be non-empty. The following result obtains a tighter bound by checking against these cases.

We first show that the feasible region we need to cover in order to find an upper bound on $S_{L_1}(Q)$ is tighter than that expressed in Th. 2. In other words, solving the maximization problem over all possible pairs of popularity sets is sufficient but not necessary. In Def. 9 below, we formally define a *maximal popularity set* - which will be useful in reducing the feasible region.

Definition 9 (Maximal popularity set). Given a set P of popularity sets, a popularity set $p^* \in P$ is maximal if and only if it is not a proper subset of any other $p \in P$. That is, p^* is maximal if and only if $\nexists p \in P$ such that $p \neq p^*$ and $p \supseteq p^*$.

Th. 9 below proves that replacing a non-maximal popularity set with its maximal popularity set cannot worsen the objective function of Th. 2.

Theorem 9. Consider $p_i, p_j \in P$. Replacing p_i (resp. p_j) with its maximal popularity set $p_i^* \supseteq p_i$ (resp. $p_j^* \supseteq p_j$) in P does not decrease $|p_i \cup p_j|$.

Proof: Without loss of generality, assume that p_i is non-maximal. There exists a maximal popularity set $p_i^* \supset p_i$ such that $p_i^* \in P$. We are guaranteed that $p_i^* - p_i \neq \emptyset$.

$$\begin{aligned} |p_i^* \cup p_j| &= |(p_i \cup (p_i^* - p_i)) \cup p_j| \\ &= |(p_i \cup p_j) \cup (p_i^* - p_i)| \\ &\ge |p_i \cup p_j|. \end{aligned}$$

In the worst case, $(p_i^* - p_i)$ will be in p_j and we reach an equality. The case for p_j follows trivially due to symmetry. □

A direct implication of Th. 9 is that we can optimize Th. 2 over only maximal popularity sets instead of all possible popularity sets as shown below.

Corollary 1. The expression in Th. 2 can be rewritten as follows:

$$\begin{aligned} S_{L_1}(Q) &\le \max_{r, r' \in \mathcal{R}(D)} |popSet(r) \cup popSet(r')| \\ &= \max_{popSets\ p_i, p_j} (|p_i \cup p_j|) \\ &= \max_{maximal\ popSets\ p_i^*, p_j^*} |p_i^* \cup p_j^*| \end{aligned}$$

This result of Cor. 1 is very useful, since every maximal clique of our graph model G is a maximal popularity set.

Theorem 10. Every maximal clique of graph G generated according to Alg. 2 is a maximal popularity set over $\mathcal{R}(D)$ on query set Q.

Proof: Let C^* be a maximal clique of G, and $range_{C^*}$ represent the range-intersection of the queries in C^*. Based on Th. 4, $range_{C^*}$ is non-empty.

We place a record r inside $range_{C^*}$. Due to maximality of C^*, this intersection cannot be contained by the range-intersection of any other subset of queries of Q. We have $popSet(r) = C^*$. □

At this point, we are ready to connect the dots. We have established that there is a bijection (i.e., a one-to-one and onto mapping) between maximal popularity sets and maximal cliques (please see Th. 6 and Th. 10). Based on Cor. 1, $S_{L_1}(Q)$ can be bounded by the maximum size of the union of two maximal popularity sets, which then must be equivalent to maximizing the union of two maximal cliques of the mapped graph.

Corollary 2. The expression in Cor. 1 can be rewritten as follows:

$$\begin{aligned} S_{L_1}(Q) &\le \max_{maximal\ popSets\ p_i^*, p_j^*} |p_i^* \cup p_j^*| \\ &= \max_{maximal\ cliques\ C_i^*, C_j^*\ of\ G} |C_i^* \cup C_j^*| \end{aligned}$$

Alg. 4 shows how the bound above can be computed.

Algorithm 4 Bounding $S_{L_1}(Q)$ with $MCE(G)$

```
1:  function SENS-MCE(Query set $Q$)
2:      $G \leftarrow$GEN-GRAPH($Q$)
3:      $C^* \leftarrow MCE(G)$
4:      if $G$ contains sink vertex $S$ and $G$ has only 2 maximal
        cliques then
5:          return $|MCS(G)|$
6:      $maxUS \leftarrow 0$ //maximum union-size
7:      for Each maximal clique $C_i^* \in C^*$ do
8:          for Each maximal clique $C_j^* \in C^*$ do
9:              $s \leftarrow |C_i^* \cup C_j^*|$
10:             if $s > maxUS$ then
11:                 $maxUS \leftarrow s$
12:     return $min(|Q|, maxUS)$
```

Proof of correctness: Returns from step 12 follow directly from Cor. 2 and Th. 7. We just consider the extremity $Q = \emptyset$. G has only one maximal clique (i.e., sink S), $maxUS = 1$ and the return value is 0. Step 4 takes special care of cases where S must be involved in the solution.

Let $Q \neq \emptyset$ and assume that all optimal maximal clique pairs contain S. Since S has no edges, the other maximal-clique element of such a pair (say C^*) should satisfy $|C^*| = |MCS(G)|$ (otherwise replacing C^* with a larger maximal clique yields a better solution). On the other hand, G cannot have two such distinct maximum-sized cliques (otherwise pairing these two would produce a better solution to $|C_i \cup C_j|$ (i.e., at least $|MCS(G)| + 1$ since $C_i \neq C_j$)). We conclude that S is involved in every optimal solution only if G has only two maximal cliques (S and C^*). In this case, $S_{L_1}(Q) \geq |C^*| = |MCS(G)|$ because S does not represent an actual query.

This algorithm requires more computational effort than Alg. 3 primarily due to step 3. Alg. 3 requires finding the size of *a* maximum clique, whereas this algorithm requires generating *all* maximal cliques (which includes a maximum clique plus many other, potentially smaller, maximal cliques). Furthermore, this algorithm also requires a search over the maximal cliques after they are found (steps 7-11), plus the union operation at each iteration (step 9).

We continue from the examples in Fig. 4 and 5. We have the following maximal cliques: $C_1 = \{Q1, Q2, Q3\}$, $C_2 = \{Q3, Q4\}$, $C_3 = \{Q5\}$, $C_4 = \{S\}$. Our aim is to find two maximal cliques such that their union is maximized. We can choose, for instance C_1 and C_2, or we can choose C_1 and C_3. Both would yield the same sensitivity, which is 4.

We compare this union bound with the previous $2 \times MCS(G)$ bound. First, this bounds finds all maximal cliques rather than only a maximum clique. As a result, it can discover that there is no second maximum clique, and the sensitivity of $2 \times |C_1| = 6$ would not be possible. Second, when we choose C_1 and C_2 (the two largest cliques) we should not simply calculate $|C_1| + |C_2|$. If we do, we would be counting their intersection, i.e., $Q3$, twice. This would be another cause of overestimation. The union bound takes also this into consideration.

In addition to the above, we would like to clarify that neither of the two maximal cliques that maximize $|C_i \cup C_j|$ have to be a maximum clique. For example, if there are two maximum cliques with almost complete overlap, and two slightly smaller maximal cliques with no overlap, then the smaller maximal cliques can as well produce the highest sensitivity impact.

Algorithm 5 Improved version of Alg. 4 ($MCE(G)$ bound)

```
1:  function SENS-MCE-SORT(Query set $Q$)
2:      $G \leftarrow$GEN-GRAPH($Q$)
3:      $C^* \leftarrow MCE(G)$
4:      if $G$ contains sink vertex $s$ and $G$ has only 2 maximal cliques then
5:          return $|MCS(G)|$
6:      $C^* \leftarrow$SORT-DESCENDING-SIZE($C^*$)
7:      $maxUS \leftarrow 0$ //maximum union-size
8:      for $i = 1$ to $|C^*|$ do
9:          if $2 \times |C_i^*| \leq maxUS$ then
10:             break
11:         for $j = i + 1$ to $|C^*|$ do
12:             $s \leftarrow |C_i^* \cup C_j^*|$
13:             if $s > maxUS$ then
14:                 $maxUS \leftarrow s$
15:     return $min(|Q|, maxUS)$
```

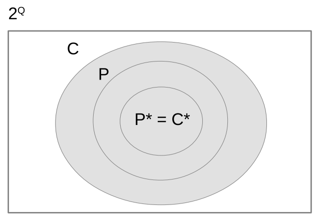

Fig. 6: Relating 2^Q, P, P^*, C and C^*

5.4 Improvements on the algorithms

We presented three bounds on $S_{L_1}(Q)$. The first bound, $S_{L_1}(Q) \leq |Q|$, is the easiest to obtain and trivial to implement. The second bound, $S_{L_1}(Q) \leq 2 \times MCS$, relies on the ability to (efficiently) find a maximum clique of a graph. The third bound, $S_{L_1}(Q) \leq max_{r,r'}|popSet(r) \cup popSet(r')|$, is the tightest bound available in the literature, and is implemented in Alg. 4 via searching over the maximal cliques of a graph. While the first bound is straightforward, our graph model serves as a means to efficiently implement the second and third bounds by reducing them to clique problems that are well-studied in graph theory. We can therefore apply the state of the art in solving clique problems to bound $S_{L_1}(Q)$ efficiently.

We report an additional observation for Alg. 4. After enumerating maximal cliques, the algorithm searches for two cliques (C_i, C_j) such that $|C_i \cup C_j|$ is maximized. Let n be the number of maximal cliques in the graph. Then, the aforementioned search is clearly $O(n^2)$. Without changing this asymptotic complexity, we note one strategy we used that significantly reduced the execution time of Alg. 4. We incorporate the strategy and provide the resulting pseudocode in Alg. 5.

In Alg. 5, our idea is to first sort the list of maximal cliques according to their size, in descending order. This sorting step is comparison-based (i.e., when sorting, we need to compare two cliques to see which one is larger in size) and comparison-based sorting can be implemented in $O(n \log n)$ time. Afterwards, the search to maximize $|C_i \cup C_j|$ begins, but when we hit a point where $2 \times |C_i|$ is no longer greater than the maximum union size found thus far (i.e., $maxUS$) the search can be stopped. Since cliques are sorted in descending order of size, we are certain at this point that $maxUS$ cannot be improved. This sorting strategy does not adversely affect computational complexity since the algorithm is $O(n^2)$ with and without it, but greatly benefits efficiency in practice. This is because many graphs have a few large maximal cliques and many smaller maximal cliques. The union of two larger cliques usually ends up providing the maximum $|C_i \cup C_j|$, which makes searching through all the smaller cliques redundant. This observation is supported by the experimental results in Sec. 6.

5.5 Limitations of the graph model

In Sec. 3.2, we concentrated on query sets containing only `COUNT` queries and showed the exact solution for $S_{L_1}(Q)$ based on the popularity set of a record r - $popSet(r)$. In this section, we investigate whether this exact solution in Th. 3 can be calculated over the graph model. This discussion will point to the limitations of our graph model.

We first summarize what we have theoretically established so far. For a query set Q, consider the following notation:

- 2^Q: power set of Q, listing every possible subset of Q as an element.
- P: set of popularity sets. Some subsets of Q may not be popularity sets, therefore $P \subseteq 2^Q$.
- P^*: set of maximal popularity sets. Some popularity sets may not be maximal, therefore $P^* \subseteq P$.
- C: cliques of G modelled from Q. Suppose there is no sink vertex S. We have $Q = V$. Cliques are fully connected subsets of V, therefore $C \subseteq 2^Q$. If V contains S, then S appears in one and only one clique: $\{S\}$ because it is a disconnected sub-graph. In this case, S represents the empty-set element of 2^Q and $C \subseteq 2^Q$ still holds.
- C^*: maximal cliques of G. Some cliques may not be maximal, therefore $C^* \subseteq C$.

Th. 6 proves that every $popSet$ is a clique. We can further say $P \subseteq C$. Th. 10 proves that every maximal clique is a maximal $popSet$. Consequently, $C^* \subseteq P^*$. We combine these two results to conclude that $C^* = P^*$. All of these results are summarized in Fig. 6.

An important question at this point is the following: $C \overset{?}{=} P$. The answer, unfortunately, is no. We can easily build a query set

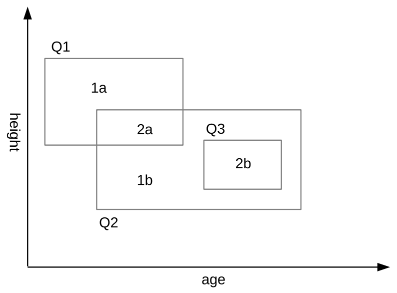

Fig. 7: Regions of queries in $\mathcal{Q}$

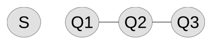

Fig. 8: Graph mapped from $\mathcal{Q}$ in Fig. 7

$\mathcal{Q}$ with a subset $s \in 2^{\mathcal{Q}}$ such that $s \in \mathcal{C}$ but $s \notin \mathcal{P}$. Consider the query set in Fig. 7 and the clique $\{Q3\}$ of the graph mapped from it in Fig. 8. For all records inside $range_{Q3}$ (the region marked 2b), $popSet(r) = \{Q3, Q4\}$. There is no record whose popularity set is exactly $\{Q3\}$. We conclude that $\mathcal{C} \not\subseteq \mathcal{P}$, and therefore, $\mathcal{P} \subset \mathcal{C}$.

This negative result is not an issue against the bound in Th. 2. According to Cor. 1, optimization domain is actually $\mathcal{P}^*$, which is equal to $\mathcal{C}^*$. Operating over $\mathcal{C}^*$ is sufficient.

The same does not hold for Th. 3. One should visit every pair (p_i, p_j) of elements from $\mathcal{P}$ to maximize $|p_i \setminus p_j| + |p_j \setminus p_i|$. Working with $\mathcal{P}^*$ does not suffice. In Fig. 7, there is only one optimal solution where $p_i = \{Q2, Q3\}$ and $p_j = \{Q1\}$. We have $S_{L_1}(\mathcal{Q}) = 3$. Notice that $p_j \notin \mathcal{P}^*$.

At first glance, one may think that the optimization in Th. 3 over $\mathcal{P}$ may be carried out over $\mathcal{C}$. Applying the same objective function over $\mathcal{C}$ hints optimal solutions as $(\{Q1, Q2\}, \{Q3\})$ or $(\{Q2, Q3\}, \{Q1\})$. The second clique solution is the exact *popSet* solution. We note that, the above case is not guaranteed. Please consider query set $\mathcal{Q}'$ in Fig. 9. The graph model of $\mathcal{Q}'$ is isomorphic to that of $\mathcal{Q}$, given in Fig. 8. Any graph-based sensitivity analysis will yield the same solution as $\mathcal{Q}$.

Applying Th. 3 on $\mathcal{Q}'$ yields $S_{L_1}(\mathcal{Q}') = 2$. This can be validated through a *popSet* analysis on Fig. 9. We express the 3 possible solutions that yield this sensitivity as pairs of subsets: (i) $(\{Q1, Q2\}, \{Q2, Q3\})$, (ii) $(\{Q1, Q2\}, \emptyset)$, (iii) $(\{Q2, Q3\}, \emptyset)$. These solutions are based on $\mathcal{P}$. However, if we instead try to apply the same optimization on $\mathcal{C}$, the solutions on $\mathcal{P}$ appear insufficient as $\mathcal{C}$ indicates that there are solutions where sensitivity is higher. Yet, all of these solutions are infeasible over $\mathcal{P}$. For example, consider $(\{Q1, Q2\}, \{Q3\})$ as a solution based on $\mathcal{C}$. In G, $\{Q3\}$ is a clique. But $\{Q3\} \notin \mathcal{P}$. Consider $(\{Q2, Q3\}, \{Q1\})$. In G, $\{Q1\}$ is a clique. But $\{Q1\} \notin \mathcal{P}$.

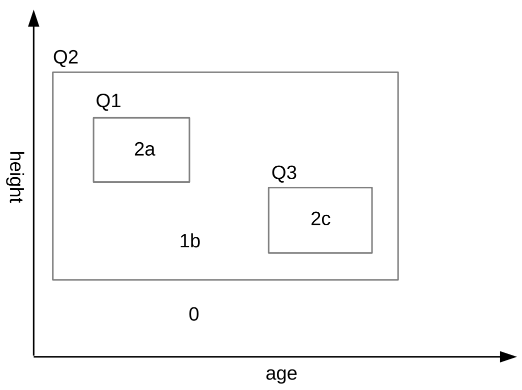

Fig. 9: Regions of queries in $\mathcal{Q}'$

We have given two sample query sets, $\mathcal{Q}$ and $\mathcal{Q}'$, with isomorphic graph models and shown that $S_{L_1}(\mathcal{Q}) = 3$ whereas $S_{L_1}(\mathcal{Q}') = 2$. We conclude that Th. 3 cannot be enforced over graph models (for an exact solution). Obviously, solving the same maximization problem over $\mathcal{C}$ instead of $\mathcal{P}$ cannot cause us to underestimate $S_{L_1}(\mathcal{Q})$. This is because $\mathcal{P} \subset \mathcal{C}$ and $\max_{p_i,p_j \in \mathcal{P}} f(p_i, p_j) \le \max_{c_i,c_j \in \mathcal{C}} f(c_i, c_j)$ trivially holds for any $f(.)$.

Naturally one would wonder how much better a bound would this be compared to Alg. 5. On the utility side, things look very bright: we are replacing the problem $\max_{c_i,c_j \in \mathcal{C}^*} |c_i \cup c_j|$ with $\max_{c_i,c_j \in \mathcal{C}} |c_i \setminus c_j| + |c_j \setminus c_i|$. Since $\mathcal{C}^* \subset \mathcal{C}$ and $|c_i \cup c_j| \ge |c_i \setminus c_j| + |c_j \setminus c_i|$ is guaranteed, the new bound is tighter than that of Alg. 5.

On the complexity side, however, the optimization domain grows from $\mathcal{C}^*$ to $\mathcal{C}$. Notice that $\mathcal{C}$ lists all subsets of each element of $\mathcal{C}^*$ and its size is much more closer to $|2^{\mathcal{Q}}|$. We consider such an optimization problem to be prohibitively complex for the extra gain in utility. That is why, we omit an algorithmic description of this solution and also exclude this bound in our experiments in Sec. 6, where typical range of $|\mathcal{Q}|$ is in hundreds.

5.6 Sensitivity for unbounded differential privacy

In Sec. 2.2 we noted a variant of differential privacy, named unbounded differential privacy (UDP), that uses the below definition of neighboring databases. The remaining definitions concerning ε-DP (i.e., Def. 2, 3 etc.) hold.

Definition 10 (Neighboring databases for UDP). Databases D, D' are called neighboring databases, if one can be obtained from the other by adding or removing a single record.

Without loss of generality, we let $D' \setminus D = r$ and $D \setminus D' = \emptyset$. That is, we obtain D' from D by adding r to D. Now, we study the sensitivity bound for UDP.

Theorem 11. For a pair of neighboring databases D, D', let $r = D' \setminus D$. Then, for a query set $\mathcal{Q}$ and its graph G, $S_{L_1}(\mathcal{Q}) \le \max_r |popSet(r)|$ and consequently $S_{L_1}(\mathcal{Q}) \le MCS(G)$.

Proof: As in the proof of Th. 2, we partition the queries in $\mathcal{Q}$ into two sets:

$popSet(r)$

$\neg popSet(r) = \mathcal{Q} \setminus popSet(r)$

Denoting $\Delta = |q(D) - q(D')|$ and starting from Def. 3:

$$S_{L_1}(\mathcal{Q}) = \max_{D,D'} \left(\sum_{\mathcal{Q}} \Delta \right) \tag{14}$$

$$= \max_{D,D'} \left(\sum_{popSet(r)} \Delta + \sum_{\neg popSet(r)} \Delta \right) \tag{15}$$

$$= \max_{D,D'} \left(\sum_{popSet(r)} \Delta + 0 \right) \tag{16}$$

$$\le \max_{r} \left(\sum_{popSet(r)} 1 \right) \tag{17}$$

$$\le \max_{r} |popSet(r)| \tag{18}$$

$$\le \max_{\text{maximal popSets } p} |p| \tag{19}$$

$$\le \max_{\text{maximal cliques } C \text{ of } G} |C| = MCS(G) \tag{20}$$

Eq. 15 breaks the summation in Eq. 14 into two terms. In Eq. 15, we observe that queries in $\neg popSet(r)$ are unaffected by the addition of r, hence their answers do not change and they do not affect $S_{L_1}(Q)$. We therefore obtain Eq. 16. Next, we use 1 as shown in Th. 1 and get Eq. 17. This can be easily converted to Eq. 18. Maximization over *popSet* sizes rules out non-maximal *popSet*s. In Eq. 19, we change the maximization domain accordingly. Finally, we use the results in Th. 6 and Th. 10 (i.e., mapping between maximal cliques and maximal *popSet*s is bijective) to obtain Eq. 20, which completes the proof. □

We now exemplify this bound. Notice that in UDP we are simply looking for a region to add a new record *r* (resp. remove an existing record *r*) such that this operation produces the highest impact over the response to *Q*. For example, in Fig. 4 placing *r* in region 3a affects Q1, Q2 and Q3. If we placed *r* in region 2d, then only 2 queries (Q3 and Q4) would be affected. Clearly, we should be searching for a region with the highest common intersection within *Q*. In *popSet* notation, this is equivalent to a maximum-sized *popSet*. Yet, a maximum-sized *popSet* is a maximum clique and its size is exactly $MCS(G)$. In Fig. 5, this maximum clique is $fQ1; Q2; Q3g$, the size of which yields the desired sensitivity bound: 3.

Let us now consider the cases where *Q* contains only `COUNT` queries. The term in Eq. 14 will be either 0 or 1. Consequently, $S_{L_1}(Q) = MCS(G)$ can be established easily. Since this result is based on a maximal clique of *G*, for the UDP case and under the condition that only `COUNT`s are queried, $S_{L_1}(Q)$ can be computed exactly based on our proposed graph model.

6 Experimental Results

We implemented Alg. 3 and 5 for estimating the sensitivity of a query set. The implementation was done almost entirely in `node.js`, and accepts statistical range queries in SQL. We parsed SQL queries using the Flora SQL parser[5]. To solve the maximum clique problem, we employed the state of the art solver $MaxCLQ$ [17] that runs on 64-bit Linux systems. For solving the maximal clique enumeration problem, we used Eppstein et al.'s solution in [14].

One of the main goals of this work is to *efficiently* approximate the sensitivity of a query set. Since computing sensitivity exactly is NP-hard (and, so are the MCP and MCEP) it is crucial to see that our system achieves its goal in reasonable time. We therefore ran various experiments to quantify the efficiency of our approach. The two most relevant parameters that affect execution time are *dimensionality* and *query set size*. Dimensionality measures how many predicates exist in a query's `WHERE` clause. Higher dimensionality requires more time to parse each query, and more time to execute Alg. 1. A larger query set adversely affects execution time in various ways. When discovering the edges of graph *G*, Alg. 2 will call Alg. 1 many more times. A larger query set will also result in a larger graph, and solving the clique problems in a larger graph is expected to take considerably more time than in a smaller graph.

We conducted experiments in two scenarios. In the first scenario, we assume a random query setting, i.e., each query is generated by randomly selecting some attributes to constrain and random constraints are placed on them. In the second scenario, we consider relational OLAP (ROLAP) queries containing popular OLAP operations, e.g., *drill-down, slice and dice*. Next, we present experimental results in these two scenarios. For each set of parameters we generated 10 query sets, and repeated all experiments 20 times for statistical significance.

6.1 Random query setting

We explore the effects of *query set size* and *average query dimensionality* on $S_{L_1}(Q)$ and the efficiency of computing $S_{L_1}(Q)$. For this, we first wrote a simple random query generator. Given a table with *t* attributes, the desired average query

5. https://github.com/godmodelabs/flora-sql-parser

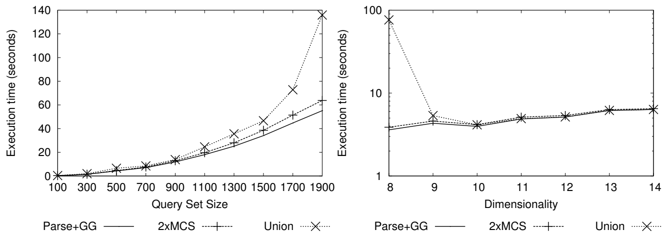

Fig. 10: Random query setting: execution time with (a) varying $|Q|$ (at $d = 9$), (b) varying d (at $|Q| = 500$)

dimensionality *d* and the query set size *jQj*, the query generator randomly generates *jQj* queries that follow the grammar in Sec. 2.4 and have average dimensionality *d*. We used $t = 15$ in all experiments.

When quantifying the effect of query set size (i.e., *jQj*) we fixed $d = 9$ and varied $jQj\ 2\ [100; 1900]$ in increments of 200. The results are given in Fig. 10(a). We draw three curves based on our measurements. The first curve (marked $Parse + GG$) depicts the time needed to parse the queries and generate their graph, i.e., Alg. 2. The second curve depicts the time needed to obtain an estimate of $S_{L_1}(Q)$ using the 2 MCS bound, i.e., Alg. 3. The third curve depicts the time needed to obtain an estimate of $S_{L_1}(Q)$ using the $max(jC_i\ [\ C_jj)$ bound, i.e., Alg. 5.

First and foremost, we see that our methods are efficient for reasonable-sized query sets. For $jQj < 800$, both bounds can be run in under 10 seconds, where query parsing and graph generation steps take the majority of the time. It takes only around 2-3 seconds to solve the clique problems and perform a search over the cliques, which is a minor overhead. For large query sets (e.g., $jQj > 1500$) we see that the $jC_i\ [\ C_jj$ bound becomes impractical, as its execution time grows rapidly. The two reasons for this are: (i) it becomes more difficult to enumerate all maximal cliques of a larger graph. (ii) After enumerating all maximal cliques, the time it takes to search for two cliques to maximize $jC_i\ [\ C_jj$ is also more time-consuming due to the increased number of cliques. On the other hand, the 2 MCS bound can be implemented in (roughly) under one minute even for large query sets (e.g., $jQj > 1700$).

When quantifying the effect of query dimensionality (i.e., *d*) we fixed $jQj = 500$ and varied dimensionality $d\ 2\ [8; 14]$ in increments of 1. The results are given in Fig. 10(b). Notice that in this graph the *y* axis is in logarithmic scale. An increase in *d* produces more constraints in a query's `WHERE` clause, and hence decreases a query's probability of intersecting with another query. As such, for large *d*, the graph becomes sparse and the clique problems can be solved efficiently. In these cases, the execution time is dominated by the parsing and graph generation steps. Whereas for denser graphs (when $d < 9$) implementing the $jC_i\ [\ C_jj$ bound becomes impractical fast.

We previously stated that Alg. 2 used for query parsing and graph generation is $O(d\ \ jQj^2)$. We see that this holds in practice. In Fig. 10(a), its curve (drawn solid) grows in a quadratic fashion, and in Fig. 10(b), its curve grows linearly.

Next, we study the $S_{L_1}(Q)$ values returned by our two bounds, 2 MCS and $jC_i\ [\ C_jj$. For the experiments above, we obtained the $S_{L_1}(Q)$ of each query set and plotted them in Fig. 11. Overall, the $jC_i\ [\ C_jj$ bound is tighter than the 2 MCS bound, as expected. However, the difference between the two bounds is small in a random query setting. This is because queries are scattered almost uniformly over the data space (D), and the maximal cliques are roughly the same size. Thus we chose to make our web application available with the 2 MCS bound. Another interesting observation is that even though *jQj* is high in both graphs, $S_{L_1}(Q)$ can be rather small. For example, for $jQj = 1500$, $S_{L_1}(Q)$ can be only 25. Therefore, naively assuming $S_{L_1}(Q) = jQj$ as a worst-case

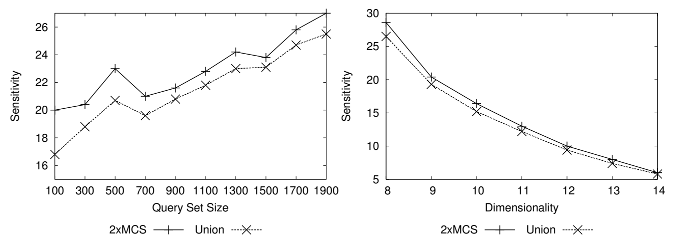

Fig. 11: Random query setting: sensitivity with (a) varying $|Q|$ (at $d = 9$), (b) varying d (at $|Q| = 500$)

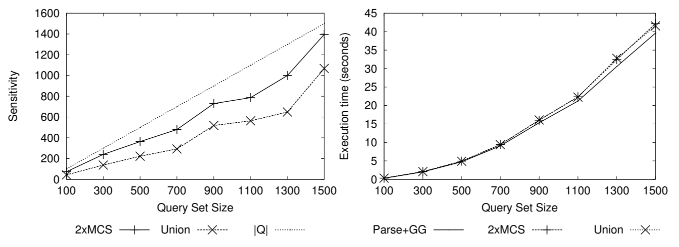

Fig. 12: ROLAP setting: (a) Sensitivity vs. $|Q|$, (b) Execution time vs. $|Q|$

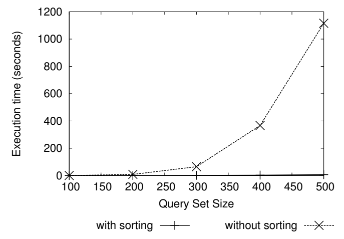

Fig. 13: Effectiveness of sorting (Alg. 5 over Alg. 4)

approximation is a *very* rough estimate. This clearly motivates using our sensitivity approximation techniques, so that a more accurate $S_{L_1}(Q)$ can be found and the noise magnitude for "-DP is not unnecessarily high.

6.2 ROLAP setting

In this scenario, we focus on popular types of ROLAP operations, e.g., drill-down, slice and dice queries [18]. In these operations, upon querying a multi-dimensional hyperrectangle, the aim is to focus deeper in some portions of the hyperrectangle by selecting one or more dimensions and stepping down in their concept hierarchy. For example, the initial query could be:

```
SELECT COUNT(*) FROM T WHERE Age BETWEEN
0 AND 100 AND Height BETWEEN 160 AND 190
```

Then, the data analyst can slice for `Age = 20` and pose the following query:

```
SELECT COUNT(*) FROM T WHERE Age = 20
AND Height BETWEEN 160 AND 190
```

Alternatively, the analyst can decide to drill-down onto the `Age` attribute by posing three separate queries for young (`AGE BETWEEN 0 AND 20`), middle-aged (`AGE BETWEEN 20 AND 50`) and elderly (`AGE BETWEEN 50 AND 100`) people. In both cases, the queries issued after the initial query have hyperrectangles that are contained by the initial query's hyperrectangle.

To obtain a query set containing ROLAP operations, we wrote a query generator that probabilistically generates queries that drill down, slice or dice the previously generated queries' hyperrectangles. We made the query generator probabilistic so that we can generate different query sets for each set of parameters.

Recall that the $S_{L_1}(Q)$ difference between our two sensitivity bounds was small in the random query setting. In the ROLAP setting, we expect the opposite: The aforementioned ROLAP operations cause some regions to be heavily queried, whereas others do not receive much attention. Therefore, we favor the creation of one large clique (say C_1), which is the maximum clique, and some other cliques that are either not as large as C_1 or have large intersections with C_1. Hence, when we apply the 2 MCS bound, we get 2 jC_1j which is large. In contrast, when we apply the jC_i [C_jj bound, we get a much smaller outcome because there does not exist a C_j, such that jC_ij jC_jj and jC_i \ C_jj is small. As a result, the potential difference between our two bounds becomes more obvious.

We experimentally illustrate this in Fig. 12(a). While the 2 MCS bound sometimes outputs $S_{L_1}(Q)$ close to jQj, the jC_i [C_jj bound asserts that $S_{L_1}(Q)$ is less than almost 30% - 40% of what the 2 MCS bound claims.

Finally, we provide the execution times for the drill-down experiments in Fig. 12(b). The execution times stay more or less the same across the drill-down experiments and random query experiments.

6.3 Effectiveness of sorting in Alg. 5

In Alg. 5 we introduced our sorting strategy and argued that it enables the jC_i [C_jj bound to be implemented much more efficiently when compared to Alg. 4. We now experimentally validate this.

We draw the execution times of Alg. 4 (without sorting) and Alg. 5 (with sorting) in Fig. 13. For this experiment, we assume the random query setting (i.e., Scenario #1). The effectiveness of Alg. 5 is clear after jQj 400: even though Alg. 5 can calculate the sensitivity bound within a couple of seconds, Alg. 4 takes around 1 minute. This is because a typical graph (built from a set of random queries) contains several tens of thousands maximal cliques, only a few of which are large sized. When sorted according to size, we find a pair of cliques that maximizes jC_i [C_jj within the first few thousand cliques (which are the few, larger cliques). On the other hand, without sorting, we need to exhaustively search all maximal cliques including those which are small in size. We therefore see that sorting improves our execution time in orders of magnitude, even for small and medium-sized query sets.

7 Related work

Differential Privacy (DP). Differential privacy was introduced by Dwork et al. in [3], and has gained significant attention ever since. For queries with real-valued outputs and functions like sums, linear algebraic functions and distance measures, the Laplace mechanism was shown to achieve DP. Later, for queries with integer-valued outputs, the geometric mechanism was proposed in [19]. A further improvement is due to McSherry et al. through the introduction of the exponential mechanism [20]. The exponential mechanism can handle queries whose responses are members of arbitrary sets, which is especially useful for mechanism design. In [21], McSherry proved the composability of multiple DP mechanisms, i.e., the *sequential* and *parallel* composition properties.

The DP definition was relaxed in many ways to increase its deployability in practical situations. The most notable relaxation is (";)-DP [22], where Def. 2 would instead be written as: $Pr[A(D)$ 2 $S]$ $e^"$ $Pr[A(D^0)$ 2 $S]$ + . (";0)-DP is equivalent to "-DP. Another relaxation is obtained by switching from the notion of *global* sensitivity (where all possible neighboring databases $(D; D^0)$ are considered, as in our work) to *local* sensitivity (where only the neighbors of a fixed database D are considered) [5]. Soria-Comas et al. argue that the above relaxations obtain gains in accuracy by breaking the DP promise [6] and instead propose a new definition called individual DP (iDP). iDP is as strong as DP with the exception that the protection mechanism operates on a fixed input database D and only its neighbors. Sensitivity computation becomes simpler and utility of the results increases but the privacy guarantees of iDP over a group of users (in contrast to individuals) is limited. Therefore, the original definition of differential privacy [3] still offers the strongest privacy protection.

Sensitivity Analysis for DP. Most closely related to our work are the sensitivity analysis and sensitivity calculation algorithms. As mentioned earlier, Xiao et al. prove in [4] that sensitivity calculation is NP-hard and provide an upper bound. Although they do not implement a solution that achieves their bound in practice, we provide solutions that realize our bounds (that are tighter than those in [4]) efficiently. [7] aims to calculate the sensitivity of queries written in relational algebra. They use constraint systems to model the behavior of relational algebra operators (e.g., selection, projection). Arapinis et al. [23] focus specifically on the sensitivity of counting queries. They find that the sensitivity of arbitrary queries are often not computable, and becomes unbounded when the query includes a join of multiple tables. However, they establish sensitivity bounds on databases that are constrained by functional dependencies and cardinality dependencies. Our work does not rely on the existence of such dependencies. [24] proposes *Fuzz*, a functional programming language with a calculus that supports the generation of differentially private functions. For functions written in this particular language, sensitivity is always well-defined and bounded. *DFuzz* [25], the successor of *Fuzz*, extends the work to a larger class of queries and functions including those whose sensitivity depends on runtime information.

This paper extends our previous work in [16] for calculating the sensitivity of SQL queries. The most notable extensions are: (i) We find tighter sensitivity bounds than those in [16], e.g., our *Union* bound and exact calculation for COUNT queries are not in [16]. (ii) We devise new algorithms that apply our bounds, e.g., we formalize that the *Union* bound can be solved via the MCEP (which requires additional theoretical study). (iii) We provide a more thorough experimental analysis.

Query Answering with DP. A fundamental task in DP is to answer a workload of statistical range queries with high utility. Above we surveyed algorithms that achieve this task via sensitivity calculation. Next we survey orthogonal approaches, e.g., those based on objective perturbation or workload perturbation.

In objective perturbation, [26] proposes that the data analysis task (e.g., the queries) is perturbed instead of its outputs. Hardt et al. [27] propose MWEM, an algorithm that combines the exponential mechanism of DP with the multiplicative weights update rule. Their idea is to select and answer those queries that are most *informative*, i.e., most incorrect in their last approximation. Li et al. [28] propose DAWA, a data-dependent algorithm that first partitions a dataset, then re-writes the query workload taking into account the partitioning, and finally answers the workload by adding appropriate noise to each partition and query answer. The matrix mechanism [29], [15] is a generic framework for answering count queries. It computes linear combinations of counts, adds noise, and then reconstructs estimates for individual counts. However, finding the optimal choice of linear combinations is computationally infeasible [30]. Finally, the low-rank mechanism [31] works by approximating a low-rank matrix of the query workload. The authors present the low-rank mechanism as a practical instantiation of the matrix mechanism.

Systems for Differentially Private Computation. Also related to our work are the practical systems for DP data analysis. These provide off-the-shelf or standalone frameworks in which the results obtained by a data analyst readily satisfy "-DP. In many cases, the DP layer is made transparent to the analyst.

The most widely known system is PINQ, which provides a querying interface built on LINQ (written in C#) [21]. Sensitivity of basic, heavily used operators (such as noisy count and noisy sum) are hardcoded for sequential composition. Airavat guarantees differential privacy for MapReduce computations [32]. GUPT uses a novel approach for managing sensitivity and the privacy budget ": It degrades privacy over time, so that utility can be better preserved [33]. In comparison, we allow the user to specify the level of privacy for each query set, and aim to maximize utility for a given privacy budget that does not change over time. These systems do not calculate sensitivity, and are therefore not comparable to our solution. However, they can be used in complementary fashion. For example, upon learning sensitivity using our system, the data analyst can set the parameters in PINQ or GUPT accordingly (e.g., by modifying the privacy budget) before obtaining noisy answers for queries that are executed in batch mode.

An important advantage of our work is its ease of integration - since we readily support SQL, our work can be directly integrated into mainstream RDBMS (that support SQL). However, methods based on histogram release, query perturbation, objective perturbation etc. all have an additional burden of intermediate computation, which may require changes to the long-established query answering strategies of RDBMSs.

8 Conclusion

Non-interactive differential privacy protects a statistical database D by adding random noise to the answers of a query set Q. Noise amount heavily depends on $S_{L_1}(Q)$, the L_1 sensitivity of Q, computing which is proven to be NP-hard. In this study, we focused on range queries and developed the tightest bounds in the literature. Our bounds are based on well studied graph problems (i.e., MCP - maximum clique problem and $MCEP$ - maximal clique enumeration problem). Utilizing the most efficient solutions to these, we showed that $S_{L_1}(Q)$ can be bounded efficiently. Even for large query sets (e.g., $jQj = 1500$) with high dimensionality, our algorithms execute in under one minute. For more practical query sets (e.g., $jQj = 500$) execution requires around 5 seconds.

References

[1] N. R. Adam and J. C. Worthmann, "Security-control methods for statistical databases: A comparative study," *ACM Comput. Surv.*, vol. 21, no. 4, pp. 515–556, Dec. 1989. [Online]. Available: http://doi.acm.org/10.1145/76894.76895

[2] C. Dwork and M. Naor, "On the difficulties of disclosure prevention in statistical databases or the case for differential privacy," *Journal of Privacy and Confidentiality*, vol. 2, no. 1, pp. 93–107, 2010. [Online]. Available: http://repository.cmu.edu/jpc/vol2/iss1/8

[3] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proceedings of the Third Conference on Theory of Cryptography*, ser. TCC'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 265–284. [Online]. Available: http://dx.doi.org/10.1007/11681878_14

[4] X. Xiao and Y. Tao, "Output perturbation with query relaxation," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 857–869, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1145/1453856.1453949

[5] K. Nissim, S. Raskhodnikova, and A. Smith, "Smooth sensitivity and sampling in private data analysis," in *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '07. New York, NY, USA: ACM, 2007, pp. 75–84. [Online]. Available: http://doi.acm.org/10.1145/1250790.1250803

[6] J. Soria-Comas, J. Domingo-Ferrer, D. Sánchez, and D. Megías, "Individual differential privacy: A utility-preserving formulation of differential privacy guarantees," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 6, pp. 1418–1429, 2017.

[7] C. Palamidessi and M. Stronati, "Differential privacy for relational algebra: improving the sensitivity bounds via constraint systems," in *10th Workshop on Quantitative Aspects of Programming Languages (QAPL)*, 2012, pp. 92–105.