Graph-based modelling of query sets for differential privacy*

Ali Inan†
Adana Science and Technology University
Department of Computer Engineering
Adana, Turkey
ainan@adanabtu.edu.tr

Mehmet Emre Gursoy
University of California at Los Angeles
Computer Science Department
Los Angeles, CA 90095
memregursoy@ucla.edu

Emir Esmerdag
Istanbul Technical University
Information Security and Cryptographic Engineering
Istanbul, Turkey
emiresmerdag@gmail.com

Yucel Saygin
Sabanci University
Faculty of Engineering and Natural Sciences
Istanbul, Turkey
ysaygin@sabanciuniv.edu

ABSTRACT

Differential privacy has gained attention from the community as *the* mechanism for privacy protection. Significant effort has focused on its application to data analysis, where statistical queries are submitted in batch and answers to these queries are perturbed with noise. The magnitude of this noise depends on the privacy parameter ε and the *sensitivity* of the query set. However, computing the sensitivity is known to be NP-hard.

In this study, we propose a method that approximates the sensitivity of a query set. Our solution builds a query-region-intersection graph. We prove that computing the maximum clique size of this graph is equivalent to bounding the sensitivity from above. Our bounds, to the best of our knowledge, are the tightest known in the literature. Our solution currently supports a limited but expressive subset of SQL queries (i.e., range queries), and almost all popular aggregate functions directly (except AVERAGE). Experimental results show the efficiency of our approach: even for large query sets (e.g., more than 2K queries over 5 attributes), by utilizing a state-of-the-art solution for the maximum clique problem, we can approximate sensitivity in under a minute.

*This research was funded by The Scientific and Technological Research Council of Turkey (TUBITAK) under grant number 114E261.

†Corresponding author

SSDBM '16 July 18–20, 2016, Budapest, Hungary

DOI:

CCS Concepts

•Security and privacy → Data anonymization and sanitization; *Privacy protections;* **•Theory of computation → Theory of database privacy and security;**

Keywords

Differential privacy, maximum clique problem, statistical database security, SQL, range queries

1. INTRODUCTION

Protecting databases against disclosure of private data of individuals through statistical analysis of the database has been studied since the early 1980s [1]. On this subject, known as statistical database security, Dwork has proven a very interesting conjecture: statistical database security cannot offer any strict guarantees to individuals like *semantic security* in cryptography [4]. In a semantically secure cryptosystem, a cipher-text does not reveal any information about the plain-text. The implications of this result 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 D. The two worlds resulting from this decision are as follows: (a) $D \leftarrow D \cup \{r\}$, (b) $D' \leftarrow D \cup \{r'\}$, where r' is the record of someone else. Differential privacy encourages participation (world (a)) by minimizing the risks Alice will be taking.

ε-differential privacy [4] offers Alice exactly the following guarantee: the probability that D and D' give the same results to a query set 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

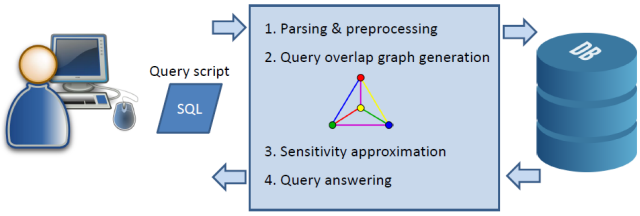

Figure 1: Work flow of the solution

sensitivity of the query set Q. This value, denoted $S_{L_1}(Q)$, is the largest effect of any single record (such as r of Alice) on the responses to Q. $S_{L_1}(Q)$ is a function of the query set and does not depend on database D.

One of the main difficulties in differential privacy is to compute $S_{L_1}(Q)$, which requires studying the outcome of Q on all possible databases D, D' differing in one record. Xiao and Tao prove in [24] that computing the sensitivity of a query set is NP-hard. This, in part, has led to the adoption of alternative approaches such as *smooth sensitivity* and *sample and aggreagate* [19], that either measure sensitivity locally (e.g., at one point) and then calibrate it to the whole database, or break the data into sample blocks, run Q on each block and then privately aggregate the results. However, it is often difficult to apply such techniques to arbitrary Q. Another approach is to assume a safe, worst-case upper bound for $S_{L_1}(Q)$ that satisfies differential privacy, but this often yields higher 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 in SQL and present algorithms that realize these bounds to compute an approximation of $S_{L_1}(Q)$. Although there has been some work in calculating sensitivity for the likes of relational algebra [20], SQL is still by far the most popular query language in 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 SQL queries, and using these ranges to convert Q to a graph. We then employ well-studied graph algorithms to approximate $S_{L_1}(Q)$.

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 will be left out to build $Q' \subseteq Q$. The interface then approximates $S^{Q'} \geq S_{L_1}(Q')$ and submits Q' to the RDBMS. Based on the privacy budget ε and the approximate sensitivity $S^{Q'}$, the query answers will be perturbed with Laplace noise drawn from $\mathcal{L}(0, S^{Q'}/\varepsilon)$ and returned to Bob. The interface currently works with statistical queries satisfying the grammar in Sec. 2.3, and databases with numeric, categorical or ordinal attributes.

$S^{Q'}$ 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`

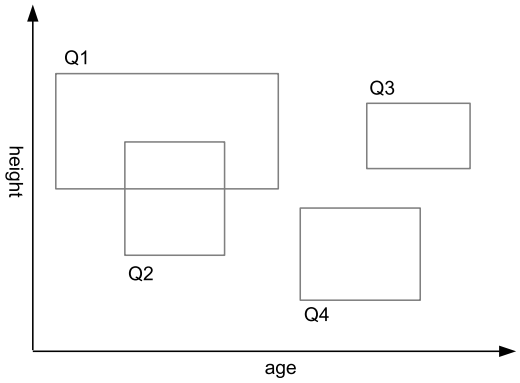

Figure 2: Regions of queries in Q'

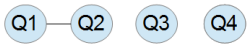

Figure 3: Graph mapped from Q'

- $Q3$: `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., each 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 set $V = Q'$, i.e., each query is represented with a vertex in G. Two vertices are connected if their query regions intersect. The resulting graph in this case is shown in Fig. 3.

We show, theoretically, that it is possible to find an upper bound on $S_{L_1}(Q')$ based solely on G. To the best of our knowledge, this upper bound improves the best-known bound in the literature, i.e., it is a tighter version of the bound presented in [24]. We show that computing this bound relies on solving the maximum clique problem (MCP) on G. Even though MCP is NP-hard (with a brute force solution that has $O(2^{|V|})$ complexity), it is one of the most heavily studied problems in computer science and there exist efficient algorithms that give an exact solution. One of the primary strengths of our approach is the exploitation of these works.

Contributions of this work can be listed as follows:

- We propose methods to map a given set of statistical queries into a graph, without requiring additional knowledge apart from the queries themselves and the domain of numerical attributes (e.g., age, height).
- We describe a novel solution for approximating the sensitivity of a query set. We theoretically prove that finding an upper bound on sensitivity is equivalent to solving the maximum clique problem on the graph.
- We utilize state-of-the-art libraries for the maximum clique problem and experimentally show that this upper bound can be computed 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. We expect integrating this implementation into commercial RDBMSs to be straightforward, so that analysts can work with the familiar SQL interface.

The rest of this paper is organized as follows. In Sec. 2.1 and Sec. 2.2, we give a brief introduction to differential privacy and the maximum clique problem. In Sec. 2.3, we list the assumptions we make on the database schema and define the types of queries that can be handled with our approach. Sec. 3 explains how a query set Q can be modelled as a graph. We bound the L_1 sensitivity $S_{L_1}(Q)$ of Q in Sec. 4. Implementation details and experimental results on the efficiency of our solution are given in Sec. 5. We review the related work in Sec. 6 and conclude in Sec. 7.

2. PRELIMINARIES

2.1 Differential Privacy

Differential privacy aims to ensure that the result of an analysis is not overly dependent on one data record. To achieve this, it conjectures that there should be a strong probability that a privacy-preserving interface produces the same result 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 $\mathcal{A}$.

In ε-DP, the user poses a set Q of queries with numeric outputs 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 *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:*

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

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

In the Laplace mechanism [4] random noise is sampled from the Laplace distribution. Scale of the distribution is determined by the privacy budget ε and $S_{L_1}(Q)$ as defined below.

DEFINITION 4 (LAPLACE MECHANISM). *Let $Lap(\sigma)$ denote a random variable sampled from the Laplace distribution with mean 0 and scale parameter σ. For queries $q : \mathcal{D} \rightarrow \mathbb{R}$, the algorithm $\mathcal{A}$ that answers each q by $\mathcal{A}(q, \mathcal{D}) = q(\mathcal{D}) + Lap(\lambda)$ is 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 more accurate results using the same ε as before.

2.2 Maximum Clique Problem

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 problem.

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, {Q2, 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 so is {Q5, Q6, Q7}. In this graph, $MCS(G) = 3$.

The maximum clique problem (MCP) has a wide range of applications, and is among the most studied combinatorial problems. Even though MCP is NP-complete [10], due to its practical relevance, there has been significant effort for finding efficient solutions. We refer the interested reader to [23] for a recent survey on algorithms for the MCP.

Although some variations of the MCP exist (e.g., listing all maximum cliques or finding a maximum weight clique in a weighted graph) our work is mostly concerned with $MCS(G)$. For this, it suffices to find one maximum clique and retrieve its size. Hence, the vast literature on solving the original MCP is directly applicable to our work.

2.3 Statistical Range Queries in SQL

Our sensitivity approximation techniques apply to non-interactive differential privacy for a restricted schema structure and a restricted subset of structured query language (SQL) queries. Details of the types of attributes and queries that are handled are given below.

We consider a database $\mathcal{D}$ containing a single d-dimensional table T with attributes $A_1, A_2, ..., A_d$. Domain of attribute A_i is denoted with $\Omega(A_i)$.

There are three requirements on the schema of T:

- 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 reducing their domain into finite, well-defined values.

- Domains of numeric attributes are normalized to the range $[0, 1)$. This requirement removes any domain dependence in 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.

Differential privacy allows only *statistical database queries*. We further limit these to queries that select a range in every dimension written in SQL. Queries of the following form are supported:

```
SELECT AGG
FROM T
WHERE pred(A_1) AND ... AND pred(A_d)
```

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. The following predicates are allowed:

- A_i `op` x, where $x \in \Omega(A_i)$ and `op` $\in \{=, >, <, \geq, \leq\}$,

- 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[1] 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 hyper-rectangle[2] in the d-dimensional domain of table T.

One can notice that all of the queries in Sec. 1 follow these conditions. However, the following queries do not:

- Q_a: `SELECT Age FROM T ...`
  Q_a is not a statistical range query since its `SELECT` clause contains an attribute name rather than an aggregate function. An answer to Q_a contains raw data, i.e., actual *age* values from the database.

- Q_b: `SELECT COUNT(*) FROM T WHERE Age > 10 AND Age > 20`
  Q_b is not valid since its `WHERE` clause contains two predicates on the same attribute, *age*.

- Q_c: `SELECT COUNT(*) FROM T WHERE Height / Age > 20`
  Q_c is not valid since its `WHERE` clause contains a predicate that is a function of two attributes.

[1]A point p in $\Omega(A_i)$ is the interval $[p, p]$

[2]We consider planes and points in d-dimensions to also be hyper-rectangles since this does not affect the correctness of our analyses.

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

Table 1: Our notation

We believe that the above is a useful subset of SQL. `COUNT` queries with rectangular ranges alone are sufficient for many important data analysis tasks such as training ID3 classifiers, building Naive Bayes models, releasing histograms and mining frequent patterns. Still, there are several SQL keywords and operators that we plan to add in the future, e.g., the `NOT IN` and `NOT BETWEEN` predicates, and the `GROUP BY` and `HAVING` clauses.

We use an SQL parser to check if given queries comply with the requirements above. Any query that does not fit into this grammar will be 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. 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. 2.3. The notation that will be used this section onwards is summarized in Table 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 $\Omega(A_i)$. 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 6. *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:*

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

Essentially, the range-intersection is the common intersection of all queries in Q_s. For example, in Fig. 4, if $Q_s = \{Q1, Q2, Q3\}$, then $range_{Q_s}$ is the area denoted 3a. If $Q_s = \{Q1, Q2, Q4\}$, then $range_{Q_s}$ is empty. If $Q_s = \{Q5, Q6\}$, then $range_{Q_s}$ is equal to $range_{Q6}$.

3.1 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. Therefore $|Q| = |V|$. The edge-set E of the graph G is constructed

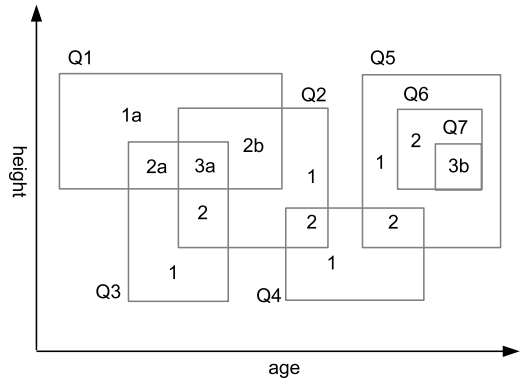

Figure 4: Regions of queries in Q

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} ← p.where[A_i]
4:         range_q^{A_i} ← q.where[A_i]
5:         if range_p^{A_i} ∩ range_q^{A_i} = ∅ 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 $q.where[A_i] = (-\infty, \infty)$ and the intersection on dimension A_i is non-empty trivially.

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

```
1: function GEN-GRAPH(Query set Q)
2:     V ← ∅
3:     for Each query q ∈ Q do
4:         V ← V ∪ {q}
5:     E ← ∅
6:     for Each query p ∈ Q do
7:         for Each query q ∈ Q, p ≠ q do
8:             if INTERSECTS(p, q) then
9:                 E ← E ∪ {(p, q)}
10:    return G(V, E)
```

The mapping algorithm that generates the actual graph is given in Alg. 2. Vertices 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.

At this point we introduce the examples in Fig. 4 and Fig. 5. Suppose that we have a query set $Q = \{Q1, ..., Q6\}$ with the ranges plotted in Fig. 4. $range_{Q1}$ and $range_{Q3}$ intersect in both dimensions (the intersection is the union of areas denoted 2a and 3a) and therefore there is an edge between $Q1$ and $Q3$ in Fig. 5. On the other hand, if we study $Q3$ and $Q4$, we observe that $range_{Q3}^{height}$ and $range_{Q4}^{height}$ intersect, but $range_{Q3}^{age}$ and $range_{Q4}^{age}$ do not. Hence, $range_{Q3} \cap range_{Q4} = \emptyset$. Consequently, in Fig. 5, there is no edge between $Q3$ and $Q4$. If we study $Q5$, $Q6$ and $Q7$, we observe that $range_{Q7} \subseteq range_{Q6} \subseteq range_{Q5}$, hence they all have a common intersection, $range_{Q7}$. Therefore in Fig. 5 they are pairwise connected to one another.

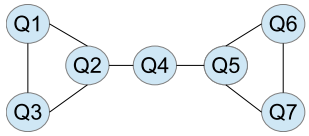

Figure 5: Graph mapped from Q

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 $O(d \times |Q|^2)$.

3.2 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 sensitivitiy of Q in Sec. 4. In this section, we present and prove these properties.

Before delving into a discussion over d-dimensional ranges, we look at the simpler case of one-dimensional spaces (1D). In 1D, a range becomes an interval. We denote an interval with $I = [l, h]$, and say the lower bound of I is $l = l(I)$ and upper bound of I is $h = h(I)$. Our first theorem is on the intersection of intervals.

THEOREM 1. *Let I, J, K be intervals. If these 3 intervals pairwise intersect, then the common intersection of the triplet should be non-empty. Formally:*

$$I \cap J \neq \emptyset, I \cap K \neq \emptyset, J \cap K \neq \emptyset \implies I \cap J \cap K \neq \emptyset$$

PROOF. Intersection of two intervals I and J is empty in the following two cases:

1. I is to the left of J: $h(I) < l(J)$ or,
2. J is to the left of I: $h(J) < l(I)$.

Therefore, $I \cap J \neq \emptyset$ implies: $l(I) \leq h(J) \wedge l(J) \leq h(I)$.

We first observe that $l(I \cap J) = \max(l(I), l(J))$. Since $I \cap K \neq \emptyset$, $l(I) \leq h(K)$. Due to $J \cap K \neq \emptyset$, $l(J) \leq h(K)$. Consequently, $l(I \cap J) = \max(l(I), l(J)) \leq h(K)$.

Similarly, $h(I \cap J) = \min(h(I), h(J))$. Since $I \cap K \neq \emptyset$, $l(K) \leq h(I)$. Due to $J \cap K \neq \emptyset$, $l(K) \leq h(J)$. Consequently, $h(K) \leq h(I \cap J) = \min(h(I), h(J))$.

Together, $l(I \cap J) \leq h(K) \wedge l(K) \leq h(I \cap J)$ implies that $(I \cap J) \cap K \neq \emptyset$ and we are done. □

Next, we generalize Th. 1 to sets of intervals.

THEOREM 2. *Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a set of n intervals. If these n intervals pairwise intersect, then their common intersection should be non-empty. Formally:*

$$\underset{1 \leq i,j \leq n, i \neq j}{\forall} I_i \cap I_j \neq \emptyset \implies \underset{1 \leq i \leq n}{\cap} I_i \neq \emptyset$$

PROOF. Consider a triplet (I_1, I_2, I_j) for $j > 2$. By Th. 1, $I_1 \cap I_2 \cap I_j \neq \emptyset$ for all $j > 2$. This means, we can remove I_1 and I_2 from the set $\mathcal{I}$ and insert $I_{1-2} = I_1 \cap I_2$. This operation allows us to reduce $\mathcal{I}$ in size: $\mathcal{I} = \{I_{1-2}, I_3, ..., I_n\}$.

Repeated application of this operation will yield $\mathcal{I} = \{I_{1-2-...-(n-1)}, I_n\}$, where $I_{1-2-...-(n-1)}$ is the non-empty interval $\bigcap_{1 \leq i \leq n-1} I_i$ and the pair of intervals $(I_{1-2-...n-1}, I_n)$ intersect. Consequently, $\bigcap_{1 \leq i \leq n} I_i \neq \emptyset$. □

Having shown these properties for 1-dimensional intervals, we are now ready to extend them to d-dimensional ranges and draw conclusions on the graph G.

THEOREM 3. *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:*

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

PROOF. Consider two vertices p, q of G. If $(p, q) \in E$, then $range_p \cap range_q \neq \emptyset$. Intersecting ranges imply intersection on every dimension i. Therefore $range_p^i \cap range_q^i \neq \emptyset$.

By definition of cliques, all vertices of the clique Q_s are connected. Consequently, for all $p, q \in Q_s$ and every dimension i, $range_p^i \cap range_q^i \neq \emptyset$. Here, applying Th. 2 yields that on dimension i, the range-intersection is non-empty: $range_{Q_s}^i \neq \emptyset$.

Since $range_{Q_s}^i \neq \emptyset$ on all dimensions i, we conclude that $range_{Q_s} \neq \emptyset$. □

THEOREM 4. *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:*

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

PROOF. For any $p, q \in Q_s$, we have $range_{Q_s} \subseteq range_p$ and $range_{Q_s} \subseteq range_q$. Consequently, $range_p \cap range_q \supseteq range_{Q_s} \neq \emptyset$. By construction of the graph G in Alg. 2, this implies that $(p, q) \in E$.

Since $Q_s \subseteq V$ and $(p, q) \in E$ for any $p, q \in Q_s$, Q_s is a clique of graph G by definition. □

Together, Th. 3 and Th. 4 indicate the equivalance 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 $\{Q5, Q6, Q7\}$ are cliques in the graph, and their range-intersections are 3a and 3b 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 $(3a \cup 2b)$. Furthermore, $\{Q4, Q5\}$ is a clique with a range-intersection denoted 2 in Fig. 4. Continuing in this fashion, one can see that all cliques have a non-empty common intersection.

4. BOUNDING SENSITIVITY

Differential privacy defines the sensitivity of a query set over all neighboring databases $\mathcal{D}$ and $\mathcal{D}'$, where each differ from the other in only one record (please see Def. 1 and Def. 3). Let $\mathcal{T}$ be the set of records common to $\mathcal{D}$ and $\mathcal{D}'$ and, r and r' denote the records that are different. Specifically, $\mathcal{T} = \mathcal{D} \cap \mathcal{D}'$, $r = \mathcal{D} - \mathcal{D}'$ and $r' = \mathcal{D}' - \mathcal{D}$.

We start our analysis with a critical observation. If the assumptions given in Sec. 2.3 on attribute domains $\Omega(A_i)$ hold and the queries q fit into the grammar, the effect of a single record change (i.e., $r \to r'$) on the query q can be bounded easily.

THEOREM 5. *For any query q and any neighboring databases $\mathcal{D}, \mathcal{D}'$; under the assumptions of Sec. 2.3, $|q(\mathcal{D}) - q(\mathcal{D}')| \leq 1$.*

PROOF. q may be a `COUNT`, a `SUM` or a `MIN/MAX` query. Each of these cases is covered independently below.

`COUNT` queries:

$$\begin{aligned} |q(\mathcal{D}) - q(\mathcal{D}')| &= |q(\mathcal{T} \cup \{r\}) - q(\mathcal{T} \cup \{r'\})| \\ &= |q(\mathcal{T}) + q(\{r\}) - q(\mathcal{T}) - q(\{r'\})| \\ &= |q(\{r\}) - q(\{r'\})| \leq 1. \end{aligned}$$

If $r \in range_q$, $q(\{r\})$ is 1, otherwise it is 0. The same holds for r'. Therefore, there are four possible combinations based on whether $r \in range_q$ and $r' \in range_q$. For all these combinations, it easy to see that $|q(\{r\}) - q(\{r'\})|$ is either 0 or 1.

`SUM` queries:

Similar to above, we have $|q(\mathcal{D}) - q(\mathcal{D}')| = |q(\{r\}) - q(\{r'\})| \leq 1$. Notice that $\Omega(A_i)$ is normalized to [0, 1). Consequently, if $r \in range_q$, $q(\{r\}) \in [0, 1)$, 0 otherwise. The same holds for r'.

`MIN/MAX` queries:

For this case, we observe that $q(.) \in [0, 1)$ due to domain normalization. Therefore, $|q(\mathcal{D}) - q(\mathcal{D}')| \leq 1$ holds. □

Notice that we have bounded in Th. 5, the summation term in the sensitivity definition (please see Def. 3). A straightforward application of this bound gives a crude upper bound on $S_{L_1}(Q)$.

THEOREM 6. *For any query set Q, under the assumptions of Sec. 2.3, $S_{L_1}(Q) \leq |Q|$.*

PROOF.

$$\begin{aligned} S_{L_1}(Q) &= \max_{\mathcal{D}, \mathcal{D}'} \left(\sum_{q \in Q} |q(\mathcal{D}) - q(\mathcal{D}')|\right) \\ &\leq \max_{\mathcal{D}, \mathcal{D}'} \left(\sum_{q \in Q} 1\right) \\ &\leq |Q|. \end{aligned}$$

□

THEOREM 7. *For any query set Q, under the assumptions of Sec. 2.3, $S_{L_1}(Q) \leq 2 \times MCS(G)$, where G is the graph generated according to Alg. 2 and $MCS(G)$ represents the size of the maximum clique of G.*

PROOF. Let $r = \mathcal{D} - \mathcal{D}'$ and $r' = \mathcal{D}' - \mathcal{D}$. We partition queries q in Q into 4 mutually exclusive and collectively exhaustive sets based on whether $r \in range_q$ and $r' \in range_q$. These cases are as follows:

- $Q_{r!r'} = \{q \in Q : r \in range_q \wedge r' \notin range_q\}$.
- $Q_{!rr'} = \{q \in Q : r \notin range_q \wedge r' \in range_q\}$.
- $Q_{rr'} = \{q \in Q : r \in range_q \wedge r' \in range_q\}$.

- $Q_{!r!r'} = \{q \in Q : r \notin range_q \wedge r' \notin range_q\}$.

If we denote the term $|q(\mathcal{D}) - q(\mathcal{D}')|$ with Δ, sensitivity will be calculated as follows.

$$S_{L_1}(Q) = \max_{\mathcal{D},\mathcal{D}'} \left(\sum_{Q} \Delta \right) \quad (1)$$

$$= \max_{\mathcal{D},\mathcal{D}'} \left(\sum_{Q_{r!r'}} \Delta + \sum_{Q_{!rr'}} \Delta + \sum_{Q_{rr'}} \Delta + \sum_{Q_{!r!r'}} \Delta \right) \quad (2)$$

$$= \max_{\mathcal{D},\mathcal{D}'} \left(\sum_{Q_{r!r'}} \Delta + \sum_{Q_{!rr'}} \Delta + \sum_{Q_{rr'}} \Delta + 0 \right) \quad (3)$$

$$\leq \max_{\mathcal{D},\mathcal{D}'} \left(\sum_{Q_{r!r'}} \Delta + \sum_{Q_{!rr'}} \Delta + \sum_{Q_{rr'}} \Delta + \sum_{Q_{rr'}} \Delta \right) \quad (4)$$

$$\leq \max_{\mathcal{D},\mathcal{D}'} \left(\sum_{Q_r} \Delta + \sum_{Q_{r'}} \Delta \right) \quad (5)$$

$$\leq \max_{\mathcal{D},\mathcal{D}'} \left(|Q_r| + |Q_{r'}| \right) \quad (6)$$

$$\leq \max_{\mathcal{D},\mathcal{D}'} \left(MCS(G) + MCS(G) \right) \quad (7)$$

$$\leq 2 \times MCS(G) \quad (8)$$

Here, Eq. 2 opens up the sum on the 4 exclusive and exhaustive cases. Eq. 3 sets one sum to 0. If a query is not affected by r and r', then its answer on $\mathcal{D}$ and $\mathcal{D}'$ should be equal (since it depends on only $\mathcal{D} \cap \mathcal{D}'$).

In Eq. 4, we introduce another sum to the expression and obtain an inequality. Eq. 5 merges $Q_{r!r'}$ (i.e., queries affected by r but not r') and $Q_{rr'}$ (i.e., queries affected by both r and r') into Q_r (i.e., queries affected by r). Similarly, $Q_{!rr'}$ and $Q_{rr'}$ are merged into $Q_{r'}$.

Th. 5 proves that $\Delta \leq 1$. Eq. 6 uses this to simplify the sum. In Eq. 7, we observe that $r \in range_{Q_r}$ and $r' \in range_{Q_{r'}}$. Based on Th. 3, both Q_r and $Q_{r'}$ should be cliques of G. The largest possible size of Q_r or $Q_{r'}$ is the maximum-clique-size $MCS(G)$ of G.

Recall that Th. 3 works only for cliques of size 2 or more. For the sake of completeness, we should also cover the cases where $MCS(G) = 1$ (i.e., none of the queries in Q intersect). These cases are trivial, since r and r' each affect at most one (possibly distinct) query and the total effect is bounded by $2 = 2 \times MCS(G)$ from above. □

Together, Th. 6 and Th. 7 gives the tightest bound available in the literature on the sensitivity of a query set: $S_{L_1}(Q) \leq \min(|Q|, 2 \times MCS(G))$. Using this bound, we give the following simple algorithm for approximating $S_{L_1}(Q)$.

Algorithm 3 Approximating $S_{L_1}(Q)$

```
1: function APPROX-SENS(Query set Q)
2:     G ←GEN-GRAPH(Q)
3:     MCS ← MCS(G)
4:     return min(|Q|, 2 × MCS)
```

Alg. 3 is very simple but expresses the main advantages of our approach: The graph G encodes all necessary information for bounding the sensitivity of the queries in Q. 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. Recall that at the beginning of this section, we presented a change in one record 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 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 cliques are $C_1 = \{Q1, Q2, Q3\}$ and $C_2 = \{Q5, Q6, Q7\}$. Therefore, if we place $r \in range_{C_1}$ (i.e., r is in range-intersection of $\{Q1, Q2, Q3\}$, denoted 3a) and $r' \in range_{C_2}$ (i.e., r' is in range-intersection of $\{Q4, Q5, Q6\}$, denoted 3b) the removal of r will affect three queries and the addition of r' will affect three queries. The actual sensitivity in this case is at most $3 + 3 = 6$, which we find exactly using Alg. 3 by $2 \times 3 = 6$. Notice that (r, r') is symmetric, i.e., if we interchanged r and r' we would have obtained the same result. Also notice that a higher change in the output cannot be obtained, e.g., if we placed r in 2b and r' in 3b, the total change would be at most 5. The definition of L_1 sensitivity is concerned with the *maximum* possible change, hence this is not useful.

We finally remark that Alg. 3 is still an upper bound, and does not necessarily yield the *exact* sensitivity of a query set. To illustrate this, we study the example in Fig. 2 and Fig. 3. In this example, the maximum change is obtained when $r \in (range_{Q1} \cap range_{Q2})$ and $r' \in range_{Q3}$. The sensitivity of this query set is 3. However, Alg. 3 would approximate it as $2 \times MCS(G) = 2 \times |\{Q1, Q2\}| = 4$. As discussed earlier, an over-estimation is not a problem from a privacy point of view, but undesirable from a utility point of view.

5. IMPLEMENTATION AND EXPERIMENTS

5.1 Implementation Details

We implemented a working prototype for estimating the sensitivity of a query set using Alg. 3. The prototype is available for use via a simple web interface[3]. We plan to extend this prototype in the future and host the full version on-line as a web interface or convert it to an open source plug-in for popular commercial RDBMSs via open database connectivity (ODBC) libraries.

The current prototype was coded almost entirely in `node.js`. SQL queries are parsed with the Flora SQL parser[4], and the data is stored in a MySQL database. $MCS(G)$ is computed by a state-of-the-art maximum clique solver, $MaxCLQ$ [14][15]. $MaxCLQ$ solver runs on 64-bit Linux systems. Next, we outline the usage of the prototype, an overview of which was previously depicted in Fig 1.

Our first step is to obtain a query set from the user. Queries can be uploaded as a plain-text file or typed manually through the web interface. We instruct our SQL parser to validate the syntax of the queries and eliminate any query

[3]http://sky.sabanciuniv.edu:8000/

[4]https://github.com/godmodelabs/flora-sql-parser

that is either not syntactically valid, or does not obey the grammar in Sec. 2.3. In the prototype, the user learns which queries were thrown out through the noisy responses.

In the second step, we convert the query set to the graph model discussed in Section 3. The time complexity of this step is $O(d \times |Q|^2)$, where d is the dimensionality and $|Q|$ is the query set size (only valid queries are relevant).

Then we approximate $S_{L_1}(Q)$ according to Alg. 3 and display the result (together with other useful information, e.g., number of queries that were invalid) to the user.

We also support providing ε-DP answers to valid queries using the Laplace Mechanism. This step is optional, and requires a MySQL database connection to be established (via the web interface) before the queries are posed. The user also needs to specify the level of privacy, i.e., ε, before the queries can be answered.

An interesting aspect of the system is that the sensitivity of a query set can be calculated without database integration or connection. In this case, the schema of the underlying database is inferred automatically from the queries. (Here, we also need to make the implicit assumption that queries involving numerical attributes (e.g., *age*) are already normalized to [0,1)). Also, this allows the user to keep his data private. That is, the data need not be shared with our system before sensitivity approximation. We believe that this is important due to two reasons: (1) The sensitive nature of the data. The user might not feel safe disclosing his local, private database to a third-party software. Therefore we allow the user to obtain $S_{L_1}(Q)$ and use it independently, e.g., in his local data analysis task. (2) Technical difficulties. If the data is distributed, or stored in a remote location, then it might not be trivial for an everyday user to integrate his database into our system.

Due to the reasons above, we are not in conflict with those systems that sit as an additional layer of privacy between the user and the database (e.g., PINQ[16], GUPT[18]). Our system can be used for this purpose, too. On the other hand, it can also be used to complement such systems: after deciding on a Q, the user obtains $S_{L_1}(Q)$ using our system and then the privacy parameter ε in PINQ is determined accordingly.

5.2 Experimental Results

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, clique finding is also NP-hard) it is crucial to see that our system achieves these tasks 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 finding a maximum clique in a larger graph is expected to take considerably more time than in a smaller graph.

To experimentally quantify the effects of these two parameters, we first wrote a simple query generator. Given a table with t attributes, the desired average query dimensionality

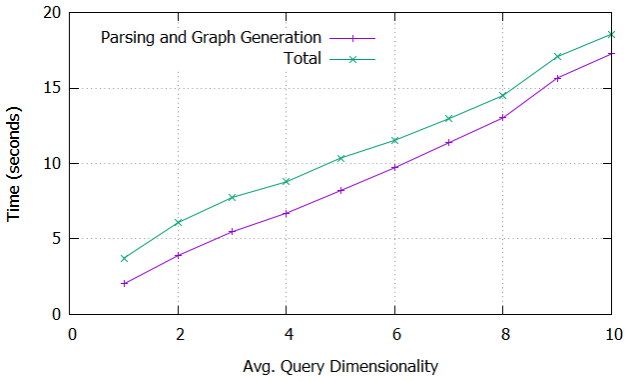

Figure 6: Execution time vs. varying dimensionality (query set size = 1000)

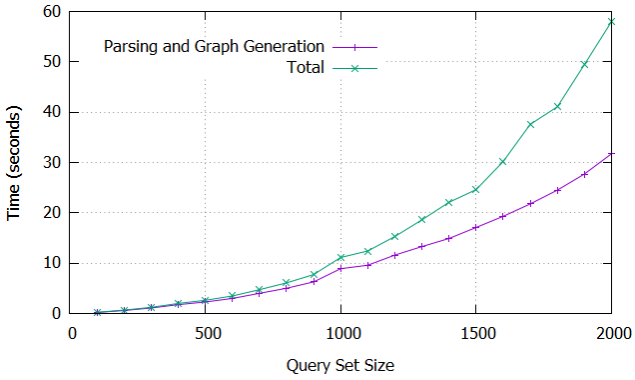

Figure 7: Execution time vs. varying query set size (dimensionality = 5)

d and the query set size s, the query generator randomly generates s queries that follow the grammar in Sec. 2.3 and have average dimensionality d. We used $t = 15$. In the first set of experiments, we fixed $s = 1000$ and generated 20 query sets for each $d = 1, 2, ..., 10$ in increments of 1. In the second set of experiments, we fixed $d = 5$ and generated 20 query sets for each $s = 100, 200, ..., 2000$ in increments of 100.

We measure the execution time of the two steps in Alg. 3 separately. We denote the time spent on parsing queries and generating the graph by "Parsing and Graph Generation" in Fig. 6 and Fig. 7. The total execution time of the algorithm, i.e., from query submission to obtaining the approximate sensitivity, is denoted by "Total" in the figures.

The results are given in Fig. 6 and Fig. 7. Each experiment was repeated 10 times for statistical significance. We draw several conclusions from these results. First, our methods are efficient. For 2000 queries, our system is able to return an answer in less than a minute, which we believe is a reasonable time frame. In more practical and reasonable scenarios (e.g., where the user has 400-500 queries with dimensionality 5) an answer can be returned in under 3 seconds. This is a very minor overhead for solving an NP-hard problem. This result is also thanks to MaxCLQ, our maximum clique solver. The time it takes to find a maximum clique of a graph with less than 1000 vertices seems to be negligible (see Fig. 7). For

1000 queries, it takes around 2-3 seconds (see the difference between the two curves in Fig. 6). Solving the maximum clique problem starts being a significant overhead only after the query set size reaches 1300. In other cases, query parsing and graph generation seem to dominate execution time.

In addition, we have stated earlier that the time complexity of query parsing and graph generation is $O(d \times |Q|^2)$. This seems to hold in practice. The relationship between execution time and dimensionality is linear in Fig. 6, as expected. The relationship between execution time and query set size seems to be superlinear (possibly quadratic) in Fig. 7, which is also expected.

6. RELATED WORK

Differential privacy (DP) was introduced by Dwork in [4], and has gained significant attention ever since. In DP, the data analyst poses a data analysis task once and uses his privacy budget ε to obtain noisy, private answers. Our study focuses on cases where the data analysis task consists of SQL queries, but in general, more complex analyses and algorithms can also be run (e.g., machine learning, data release algorithms).

We first discuss the most influential advances in DP. For queries with real-valued outputs, the Laplace mechanism was shown to achieve DP [4]. Even though this result was initially only for count queries, Dwork et al. extended the Laplace mechanism to functions like sums, linear algebraic functions and distance measures [6]. Later, for queries with integer-valued outputs, the geometric mechanism was proposed in [8]. A further improvement is due to McSherry et al. through the introduction of the exponential mechanism [17]. The exponential mechanism can handle queries whose responses are members of arbitrary sets, which is especially useful for mechanism design. In [16], 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 (ε, δ)-DP [5], where Def. 2 would instead be written as: $Pr[\mathcal{A}(\mathcal{D}) \in S] \leq e^{\varepsilon} \times Pr[\mathcal{A}(\mathcal{D}') \in S] + \delta$. $(\varepsilon, 0)$-DP is equivalent to ε-DP. Another relaxation is obtained by switching from the notion of *global* sensitivity (where all possible neighboring databases (D, D') are considered, as in our work) to *local* sensitivity (where only the neighbors of a fixed DB D are considered) [19].

A fundamental task that has received much effort in DP is to answer statistical range queries with high utility. A prominent method is *output perturbation*. Xiao et al. show in [24] how new count queries can be answered privately, using responses to previous queries. Their solution is based on a histogram approach that partitions the data space into non-overlapping subspaces. This study also proves that computing $S_{L_1}(Q)$ is NP-hard, and provides an upper bound on $S_{L_1}(Q)$.Their result is similar to our bound, however we would like to emphasize that our query model works also for `MIN`, `MAX`, `SUM` queries, and the bounds we provide are tighter than those in [24]. Additionally, even though [24] does not implement a solution to achieve their bound in practice, we provide a solution that realizes our bound efficiently.

Objective perturbation is an alternative to output perturbation. In objective perturbation, [3] proposes that the data analysis task (e.g., the queries) is perturbed, instead of the queries' outputs, to satisfy privacy. This is orthogonal to our approach. Furthermore, efforts have also focused specifically on answering count queries and linear queries. These efforts do not provide an interface that can directly be integrated into mainstream RDBMS that support SQL, and some efforts assume subsets of the query and data models we present. Among notable works in this domain are the MWEM [9] and DAWA [11] algorithms, and the matrix [12, 13] and low-rank mechanisms [25].

Also related to our work are practical studies that propose differentially private systems and languages, which can be employed for private data analysis. The PINQ system provides a querying interface built on LINQ of the C# language [16]. PINQ is a purely compositional DP interface. 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 [22]. GUPT uses a novel approach for managing sensitivity and the privacy budget ε: It degrades privacy over time, so that utility can be better preserved [18]. 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 compute $S_{L_1}(Q)$, and are not comparable to our solution. However, they can be used in complementary fashion. For example, upon learning $S_{L_1}(Q)$ 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. In addition, we refer the reader to [2] for a survey on using programming language techniques to formally verify that a given system satisfies DP.

Finally, we study the related work on sensitivity calculation for DP. As mentioned earlier, among the results of [24] is an upper bound on $S_{L_1}(Q)$ for count queries. [20] 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). [21] proposes *Fuzz*, a functional programming language with a calculus that supports the generation of differentially private functions. For functions written in this particular language, they show that sensitivity is always well-defined and bounded. *DFuzz* [7], the successor of *Fuzz*, extends the work to a larger class of queries and functions including those whose sensitivity depends on runtime information.

7. CONCLUSION

The primary difficulty of applying non-interactive differential privacy to an analysis task is to compute the sensitivity of a query set Q. In this study, we work with a restricted yet very expressive subset of statistical range queries in SQL. We model Q as a graph whose vertices are the queries in Q. Edges of the graph indicate that the ranges of the connected queries intersect. We prove that $S_{L_1}(Q)$ is less than or equal to the minimum of $|Q|$ and $2 \times MCS(G)$, where $MCS(G)$ is the maximum clique size of the graph mapped from Q. These bounds are the tightest available in the literature. Computing $MCS(G)$ can be done efficiently due to existing work on the maximum clique problem. Empirical analysis on complex query sets (e.g., 2K queries over 5 attributes) show the efficiency of our approach, as the result can be computed in under a minute.

In future work, we plan to improve our sensitivity bounds further and also aim for an exact solution. These will likely

require additional constraints on the data and query model. Another alternative direction, that is more of practical value, will be strengthening the prototype implementation to support commercial RDBMSs in a more trivial way - such as an ODBC connection.

8. REFERENCES

[1] N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.*, 21(4):515–556, Dec. 1989.

[2] G. Barthe, M. Gaboardi, J. Hsu, and B. Pierce. Programming language techniques for differential privacy. *ACM SIGLOG News*, 3(1):34–53, Feb. 2016.

[3] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *The Journal of Machine Learning Research*, 12:1069–1109, 2011.

[4] C. Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12, Venice, Italy, July 2006. Springer Verlag.

[5] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology (EUROCRYPT 2006)*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503, Saint Petersburg, Russia, May 2006. Springer Verlag.

[6] 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*, TCC'06, pages 265–284, Berlin, Heidelberg, 2006. Springer-Verlag.

[7] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. *SIGPLAN Not.*, 48(1):357–370, Jan. 2013.

[8] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.

[9] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *Advances in Neural Information Processing Systems*, pages 2339–2347, 2012.

[10] R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[11] C. Li, M. Hay, G. Miklau, and Y. Wang. A data-and workload-aware algorithm for range queries under differential privacy. *Proceedings of the VLDB Endowment*, 7(5):341–352, 2014.

[12] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 123–134. ACM, 2010.

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

[14] C.-M. Li and Z. Quan. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 344–351. IEEE, 2010.

[15] C. M. Li and Z. Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, volume 10, pages 128–133, 2010.

[16] F. McSherry. Privacy integrated queries. *Communications of the ACM*, September 2010.

[17] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Providence, RI, October 2007. IEEE.

[18] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. Gupt: Privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 349–360, New York, NY, USA, 2012. ACM.

[19] 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*, STOC '07, pages 75–84, New York, NY, USA, 2007. ACM.

[20] 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)*, pages 92–105, 2012.

[21] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. *ACM SIGPLAN Notices*, 45(9):157–168, 2010.

[22] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[23] Q. Wu and J.-K. Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015.

[24] X. Xiao and Y. Tao. Output perturbation with query relaxation. *Proc. VLDB Endow.*, 1(1):857–869, Aug. 2008.

[25] G. Yuan, Z. Zhang, M. Winslett, X. Xiao, Y. Yang, and Z. Hao. Low-rank mechanism: optimizing batch queries under differential privacy. *Proceedings of the VLDB Endowment*, 5(11):1352–1363, 2012.