Output Perturbation with Query Relaxation

Xiaokui Xiao   Yufei Tao
Department of Computer Science and Engineering
Chinese University of Hong Kong
Sha Tin, New Territories, Hong Kong
{xkxiao, taoyf}@cse.cuhk.edu.hk

ABSTRACT

Given a dataset containing sensitive personal information, a statistical database answers aggregate queries in a manner that preserves individual privacy. We consider the problem of constructing a statistical database using *output perturbation*, which protects privacy by injecting a small noise into each query result. We show that the state-of-the-art approach, *ϵ-differential privacy*, suffers from two severe deficiencies: it (i) incurs prohibitive computation overhead, and (ii) can answer only a limited number of queries, after which the statistical database has to be shut down. To remedy the problem, we develop a new technique that enforces ϵ-different privacy with economical cost. Our technique also incorporates a *query relaxation* mechanism, which removes the restriction on the number of permissible queries. The effectiveness and efficiency of our solution are verified through experiments with real data.

1. INTRODUCTION

The evolution of information technology has enabled an organization (e.g., hospitals, retailers) to collect large volumes of sensitive personal data (e.g., medical records, transaction history), which is usually referred to as *microdata*. To facilitate research, these organizations often need to provide public access to their microdata, which, however, may pose a risk to individual privacy. For example, assume that the Census Bureau maintains an online database for answering count queries on the microdata T in Table 1, which contains three columns, *Age*, *Zipcode*, and *Income* (*Name* is included to facilitate row referencing). Consider an adversary who knows the age 20 and zipcode 15000 of Alice, and the fact that Alice is involved in T. To infer the income of Alice, the adversary may issue the following two queries q_0 and q_0':

q_0: SELECT COUNT(*) FROM T
WHERE *Age* $\in$ [20, 20] AND *Zipcode* $\in$ [15k, 15k]
AND *Income* $\in$ [80k, $+\infty$)

q_0': SELECT COUNT(*) FROM T
WHERE *Age* $\in$ [20, 20] AND *Zipcode* $\in$ [15k, 15k]
AND *Income* $\in$ $(-\infty$, 80k)

VLDB '08, August 24-30, 2008, Auckland, New Zealand

Name	Age	Zipcode	Income
Alice	20	15000	85k
Bob	25	52000	32k
Cathy	33	41000	25k
David	38	23000	37k
Eva	44	26000	43k
Frank	47	18000	65k
George	53	31000	28k
Helen	61	35000	54k

Table 1: The microdata

The answers of q_0 and q_0' are 1 and 0, respectively. Once these results are returned, the adversary can assert that Alice's income must be above 80k, a close guess of Alice's real salary 85k.

The above problem motivates *statistical databases*, which answer counting queries without leaking individuals' privacy. An effective approach is *output perturbation* [2, 6, 11, 13], which works by injecting a small random noise into each query result. For queries that pinpoint sensitive information (e.g., q_0 and q_0'), their answers are dominated by noise; hence, privacy is preserved. On the other hand, the noise has little effect on queries that retrieve high-level statistics (e.g., find the number of people earning more than 30k), since they usually have large results.

Numerous output perturbation techniques are available in the statistics literature (see [2] and the references therein). Those techniques, however, are not based on a rigorous definition of privacy [12]. To overcome this defect, Dinur and Nissim [11] develop a principle called *ϵ-differential privacy* (to be elaborated in Section 2), and employs it to avoid queries that can reveal sensitive information. Specifically, let Q be the set of previously answered queries. Given a new query q, the database determines whether $\{q\} \cup Q$ violates ϵ-differential privacy. If yes, q is rejected; otherwise, the database reports a noisy result. As proved in [11], this approach guarantees that an adversary can recover any sensitive information with very low probability, even if s/he has audited the results of all the queries in history.

1.1 Motivation

Despite being the state of the art, ϵ-differential privacy has two drawbacks that severely reduce its practical applicability. First, somewhat surprisingly, there is no existing solution for checking ϵ-differential privacy. As detailed in the next section, the difficulty stems from the computation of the so-called L_1 *sensitivity*, which is a crucial component in verifying ϵ-differential privacy. The best efforts are due to Dinur et al. [13], who point out several special cases where L_1 sensitivity can be calculated. Similar attempts have also been made in [5, 20, 24]. Unfortunately, the calculation problem in general is still open. In other words, currently ϵ-differential privacy is virtually inapplicable when arbitrary queries are allowed.

The second defect of ϵ-differential privacy also exists in all the previous output perturbation solutions. Specifically, when the database denies a query, it simply returns nothing. This incurs rather negative user experience, because a legitimate user would have to spend a long time trying different queries before getting an answer. Even worse, ϵ-differential privacy supports only a finite number of queries [11]. In other words, after a period of time, the statistical database will have to go offline, and all future queries are directly refused.

In fact, for a denied query q, it is possible to return a useful *synthetic answer*, which is synthesized from the reported answers of the past queries. To illustrate, assume that the database reported an answer a_1 for query q_1:

q_1: SELECT COUNT(*) FROM T
WHERE *Age* $\in [20, 50]$ AND *Income* $\in$ [40k, 70k]

and now receives a query q_1':

q_1': SELECT COUNT(*) FROM T
WHERE *Age* $\in [20, 51]$ AND *Income* $\in$ [40k, 70k]

If q_1' needs to be denied for privacy preservation, we may still return a_1 to the user, along with the definition of q_1 (so that the user knows a_1 is the result of query q_1 that *relaxes* her/his original query q_1'). Since the predicates in q_1 and q_1' are similar, the answer a_1 would still be useful to the user. We refer to the process as *relaxation*.

In general, relaxation may combine the results of multiple queries. To illustrate, consider:

q_2: SELECT COUNT(*) FROM T
WHERE *Age* $\in [30, 69]$ AND *Income* $\in [0, 39999]$

q_3: SELECT COUNT(*) FROM T
WHERE *Age* $\in [30, 69]$ AND *Income* $\in [40000, 99999]$

q_3': SELECT COUNT(*) FROM T
WHERE *Age* $\in [30, 69]$ AND *Income* $\in [0, 99999]$

The exact result of q_3' equals the sum of those of q_2 and q_3. Assume that the database has returned a result a_2 (a_3) for q_2 (q_3), but denies q_3'. In this case, we may report a synthetic answer $a_2 + a_3$ for q_3'. Note that the answer is approximate, because a_2 and a_3 are noisy. Furthermore, returning the synthetic answer does not compromise any privacy guarantee. This is because both queries q_2 and q_3, as well as their reported results a_2 and a_3, are already public knowledge. Anything derived *solely* from such knowledge is also public knowledge.

It is worth pointing out that the meaning of query relaxation in our context is drastically different from its counterpart in relational databases [19, 27]. Specifically, in [19, 27], when an SQL query returns an empty result, relaxation performs the smallest modification to the query predicates in order to retrieve at least one tuple. The solutions in [19, 27] cannot be adapted to our circumstances.

1.2 Contributions

This paper proposes a novel output-perturbation solution based on an in-depth study of the algorithmic aspects of ϵ-differential privacy. First, we prove, for the first time, that exact computation of L_1 sensitivity is NP-hard. Recall that L_1 sensitivity is required in checking ϵ-differential privacy. Thus, the NP-hardness result rules out the existence of any algorithm for verifying ϵ-differential privacy efficiently.

Fortunately, it is possible to efficiently calculate a 2-approximate upper bound of the L_1 sensitivity. This result leads to a fast approach that verifies ϵ-differential privacy in a safe, conservative, manner. Specifically, when ϵ-differential privacy does not hold, our solution always correctly indicates so, thus guaranteeing that privacy breach can never happen.

Another salient feature of the proposed technique is that it incorporates an effective query relaxation mechanism, to provide useful answers to the denied queries. This remedies the common defect of all the previous output-perturbation solutions (mentioned in Section 1.1), because now a user no longer needs to go through the annoying process of modifying her/his query repetitively. Instead, s/he immediately obtains a similar query suggested by the database, together with the query's answer. We perform extensive experiments to evaluate our algorithms, and confirm their effectiveness and efficiency in practice.

The rest of the paper is organized as follows. Section 2 reviews ϵ-differential privacy and its related concepts. Section 3 studies the computation of L_1 sensitivity, and presents our conservative method for verifying ϵ-differential privacy. Section 4 elaborates the details of query relaxation. Section 5 contains an experimental evaluation. Section 6 reviews the previous work related to ours. Finally, Section 7 concludes the paper with directions for future work.

2. PRELIMINARIES

Let T be a microdata table, which contains d attributes A_1, ..., A_d with finite and discrete domains. We aim to support queries of the form

SELECT COUNT(*) FROM T
WHERE $pred(A_1)$ AND ... AND $pred(A_d)$

such that $pred(A_i)$ has the format

$$A_i = * \text{ or } A_i \in [x_i, y_i],$$

where x_i and y_i are two values in the domain of A_i[1]. We consider count queries, because of their imperative roles in various data analysis tasks, including OLAP, association rule mining, decision tree learning, etc.

Given a query q, we denote its real result on T as $q(T)$. To process queries in a privacy preserving manner, we adopt the output-perturbation methodology in [13] to design a statistical database $\mathcal{D}$. Specifically, given a query q, $\mathcal{D}$ returns a *perturbed answer* $q(T) + \delta$, where δ is a random variable following a *Laplace* distribution, with a probability density function

$$f(\delta) = \frac{1}{2\lambda} e^{-\frac{|\delta|}{\lambda}}. \tag{1}$$

λ is known as the *noise magnitude* of $\mathcal{D}$, and is also the expectation of $|\delta|$. We denote the perturbed answer as $q(\mathcal{D})$.

By injecting noise in the above manner, $\mathcal{D}$ ensures a strong type of privacy protection, ϵ-*differential privacy* [13]. This notion of privacy is formulated through the following definitions.

DEFINITION 1 (SIBLING TABLES). Two microdata tables T_1 and T_2 are *siblings*, if they have the same schema and cardinality, and differ in only one tuple. □

EXAMPLE 1. Let T_1 be the microdata table T in Table 1. By changing the income of Alice to another value (e.g., 30k), we obtain an alternative table T_2. T_1 and T_2 are siblings. □

[1]If A_i is categorical, we assume that there exists on A_i a total ordering, which lists the leaves of A_i's taxonomy tree [17] from left to right.

DEFINITION 2 (ϵ-DIFFERENTIAL PRIVACY [13]). Let $Q = \{q_1, ..., q_m\}$ be any subset of the queries that have been answered by $\mathcal{D}$, and $R = \{r_1, ..., r_m\}$ be a set of arbitrary real numbers. $\mathcal{D}$ ensures *ϵ-differential privacy*, if the following inequality holds for any R and any pair of sibling tables T_1 and T_2:

$$Pr\big[\forall i, q_i(\mathcal{D}) = r_i \mid \Delta_1\big] \leq e^{\epsilon} \cdot Pr\big[\forall i, q_i(\mathcal{D}) = r_i \mid \Delta_2\big],$$

where Δ_1 (Δ_2) denotes the event that T_1 (T_2) is the microdata on which $\mathcal{D}$ is constructed. □

EXAMPLE 1 (CONTINUED). Suppose that a statistical database $\mathcal{D}$ is built on T_1. Consider an adversary who tries to infer the income of Alice. Let Q be the set of queries issued by the adversary, and S_{rslt} the set of results returned by $\mathcal{D}$. If $\mathcal{D}$ ensures ϵ-differential privacy ($\epsilon \ll 1$), the adversary gains little knowledge about Alice's income, after observing S_{rslt}. To understand this, let us assume that $\mathcal{D}$ is constructed on another microdata table (e.g., T_2), where Alice's income is arbitrarily modified. By Definition 2, $\mathcal{D}$ may still return S_{rslt} as the results for the queries in Q. In particular,

$$Pr\big[\mathcal{D} \text{ returns } S_{rslt} \mid \text{Alice's income is NOT modified}\big]$$
$$\leq e^{\epsilon} \cdot Pr\big[\mathcal{D} \text{ returns } S_{rslt} \mid \text{Alice's income is modified}\big].$$

Notice that, when ϵ is small, $e^{\epsilon} \approx 1 + \epsilon$, which is close to 1. In other words, S_{rslt} provides the adversary with very little information, regarding the income of Alice. In general, a smaller ϵ leads to tighter privacy protection. □

As will be shown in Theorem 1, to decide whether $\mathcal{D}$ preserves ϵ-differential privacy, it suffices to inspect (i) the noise magnitude λ of $\mathcal{D}$, and (ii) the L_1 *sensitivity* of the queries answered by $\mathcal{D}$.

DEFINITION 3 (L_1 SENSITIVITY [13]). Given a set Q of queries, its L_1 *sensitivity* $S_{L1}(Q)$ equals:

$$S_{L1}(Q) = \max_{T_1, T_2} \left(\sum_{q \in Q} \big| q(T_1) - q(T_2) \big| \right), \tag{2}$$

where T_1 and T_2 are any two sibling microdata tables. □

EXAMPLE 2. Consider the queries q_0 and q_0' in Section 1. Let $Q = \{q_0, q_0'\}$. We will show that $S_{L1}(Q) = 2$.

Let T_1 and T_2 be any two sibling microdata tables, and $Q = \{q_0, q_0'\}$. Since T_1 and T_2 differ in one tuple, we have $\big|q_0(T_1) - q_0(T_2)\big| \leq 1$ and $\big|q_0'(T_1) - q_0'(T_2)\big| \leq 1$, which leads to $\sum_{q \in Q} \big|q(T_1) - q(T_2)\big| \leq 2$. Hence, $S_{L1}(Q) \leq 2$.

Consider that T_1 equals Table 1, and T_2 is a sibling of T_1, which changes Alice's income to 30k. We have $q_0(T_1) = 1$, $q_0(T_2) = 0$, $q_0'(T_1) = 0$, and $q_0'(T_2) = 1$. Therefore, $S_{L1}(Q) \geq |1 - 0| + |0 - 1| = 2$. Thus, $S_{L1}(Q) = 2$. □

THEOREM 1 ([13]). *A statistical database $\mathcal{D}$ ensures ϵ-differential privacy, if and only if $S_{L1}(Q) \leq \epsilon\lambda$, where λ is the noise magnitude of $\mathcal{D}$, and Q is the set of queries that have been answered by $\mathcal{D}$* [2].

Based on Theorem 1, Dwork et al. [13] propose a framework for constructing $\mathcal{D}$ as follows. Before answering any query, we choose appropriate values for λ and ϵ, which decide the query accuracy and degree of privacy protection, respectively. Then, whenever a query q is issued to $\mathcal{D}$, we inspect the set Q of queries that $\mathcal{D}$ has evaluated previously. If $S_{L1}(Q \cup \{q\}) > \epsilon\lambda$, q is denied; otherwise, $q(\mathcal{D})$ is returned as the result for q. In this way, $\mathcal{D}$ always ensures ϵ-differential privacy.

Essential to the above framework is that we must be able to decide whether $S_{L1}(Q \cup \{q\}) > \epsilon\lambda$ for any query q. This turns out to be computationally difficult, as discussed in the next section.

[2]The concept of L_1 sensitivity and Theorem 1 can be adapted to any queries (e.g., SUM, MAX, MIN) that map the microdata to real numbers. See [13] for details.

3. THE HISTOGRAM APPROACH

In Section 3.1, we prove the NP-hardness of computing $S_{L1}(Q)$, and then give a method for deriving a 2-approximate upper bound of $S_{L1}(Q)$. Section 3.2 describes a histogram approach, which enables a statistical database to process each query in an efficient and privacy preserving manner. Finally, Section 3.3 points out a limitation of output perturbation, which motivates the solutions in Section 4.

3.1 Convergence of Queries

Let $\mathcal{D}$ be a statistical database, which has a noise magnitude λ, and has answered a set Q of queries. Given a new query q, our objective is to decide if $\mathcal{D}$ still preserves ϵ-differential privacy after answering q. By Theorem 1, it suffices to verify whether $S_{L1}(Q \cup \{q\}) \leq \epsilon\lambda$. The verification turns out to be NP-hard:

LEMMA 1. *Deciding whether $S_{L1}(Q)$ is larger than a threshold is NP-hard.*

PROOF. See the appendix. □

Combining the lemma with Theorem 1 leads to:

COROLLARY 1. *Verification of ϵ-differential privacy is NP-hard.*

We thus switch our attention to calculating an upper-bound of $S_{L1}(Q \cup \{q\})$, which, as explained later, allows us to conservatively determine whether q can be answered. For this purpose, we introduce the following concepts.

DEFINITION 4 (DATA SPACE / QUERY REGION). Given T, we define its *data space* Ω as a d-dimensional space, where the i-th dimension ($1 \leq i \leq d$) is A_i. The *region* of a query q is a rectangle r in Ω such that, for any $i \in [1, d]$,

- if q has a predicate "$A_i \in [x_i, y_i]$", the projection of r on A_i equals $[x_i, y_i]$;
- otherwise (i.e., q has a predicate "$A_i = *$"), the projection of r on A_i covers all values in A_i. □

Since each A_i ($i \in [1, d]$) has a finite and discrete domain, Ω can be regarded as a set of d-dimensional points. Accordingly, the microdata T can also be viewed as a set of points.

DEFINITION 5 (POPULARITY / CONVERGENCE). Let Q be a set of queries, and R be the set containing the regions of all queries in Q. For any point p in the space Ω, its *popularity* $p(Q)$ in Q is the number of regions in R that cover p.

The *convergence* of Q, denoted as $C(Q)$, is the largest $p(Q)$ of all points $p \in \Omega$. □

EXAMPLE 3. For example, let Q consist of the queries q_1 and q_2 in Section 1.1. Figure 1 shows their regions r_1 and r_2, namely, $R = \{r_1, r_2\}$. Any point p in $r_1 \cap r_2$ has a popularity $p(Q) = 2$ in Q. If p is covered only by either r_1 or r_2, its popularity is 1. All points outside r_1 and r_2 have popularity 0. Thus, $C(Q) = 2$. □

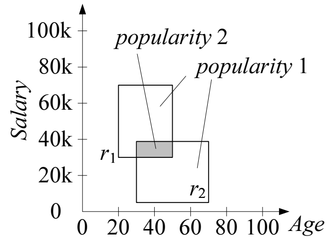


Figure 1: Popularity and convergence

Algorithm *Process* (q)
/* q is the query being answered */
ans = NULL
r = the query region of q
S_{buk} = the set of buckets in histogram $\mathcal{H}$ that intersect r
if all buckets $B \in S_{buk}$ have counters smaller than $\epsilon\lambda/2$
  $ans = q(\mathcal{D})$; $Q = Q \cup \{q\}$
  for each bucket $B \in S_{buk}$
    $B.c = B.c + 1$
    if $B.c = \epsilon\lambda/2$ and $|\mathcal{H}| < \theta$, then *Split*(B)
    /* θ is the maximum number of buckets allowed */
return ans

Figure 2: Query processing algorithm

$C(Q)$ can be used to derive a 2-approximate bound of $S_{L1}(Q)$:

LEMMA 2. *For any set Q of queries, $S_{L1}(Q) \le 2 \cdot C(Q)$.*

PROOF. Let T_1 and T_2 be two sibling microdata tables, such that $\Sigma_{q\in Q}\big|q(T_1) - q(T_2)\big| = S_{L1}(Q)$. By Definition 1, there should exist only one tuple t_1 (t_2) in T_1 (T_2) that does not appear in T_2 (T_1). Let T_3 and T_4 be two tables such that $T_3 = \{t_1\}$ and $T_4 = \{t_2\}$. We have $\Sigma_{q\in Q}\big|q(T_3) - q(T_4)\big| = S_{L1}(Q)$. For any $q \in Q$, $q(T_3)$ and $q(T_4)$ is either 0 or 1. Therefore,

$$S_{L1}(Q) = \Sigma_{q\in Q}\big|q(T_3) - q(T_4)\big| \le \Sigma_{q\in Q} q(T_3) + \Sigma_{q\in Q} q(T_4).$$

implying that either $\Sigma_{q\in Q} q(T_3)$ or $\Sigma_{q\in Q} q(T_4)$ is at least $S_{L1}(Q)/2$. Without loss of generality, assume $\Sigma_{q\in Q} q(T_3) \ge S_{L1}(Q)/2$. Let p_1 be the point in Ω whose i-th ($1 \le i \le d$) coordinate of p_1 equals $t_1[A_i]$. Let R be the set of regions of the queries in Q. By Definition 5, at most $C(Q)$ regions in R cover p_1. Hence, t_1 satisfies at most $C(Q)$ queries in Q, i.e., $\Sigma_{q\in Q} q(T_3) \le C(Q)$. Therefore, $S_{L1}(Q)/2 \le \Sigma_{q\in Q} q(T_3) \le C(Q)$, which completes the proof. □

The lemma motivates a simple approach to ensure ϵ-differential privacy. We only need to maintain the popularity $p(Q)$ for each point $p \in \Omega$. Whenever a new query q is received, we inspect the points in Ω covered by the region of q. If all of them have popularities at most $\epsilon\lambda/2$, q is answered; otherwise, q is denied. The approach, unfortunately, is impractical, since it requires keeping as many values as the points in the whole space Ω. To overcome this drawback, in the next subsection, we employ an approximation technique to monitor $C(Q \cup \{q\})$ with small space.

3.2 A Histogram Approach

Let Q be the set of queries that have been answered by $\mathcal{D}$, and R be the set of regions of those queries. We maintain a histogram $\mathcal{H}$, which partitions the data space Ω into disjoint buckets with rectangular extents. Each bucket $B \in \mathcal{H}$ is associated with a counter $B.c$, equal to the number of query regions in R intersecting B. The largest number θ of buckets in $\mathcal{H}$ is a system parameter, decided by how much space can be allocated for $\mathcal{H}$.

Apparently, any point p in B is covered by at most $B.c$ queries in Q, i.e., the popularity $p(Q)$ of p in Q is at most $B.c$. Therefore,

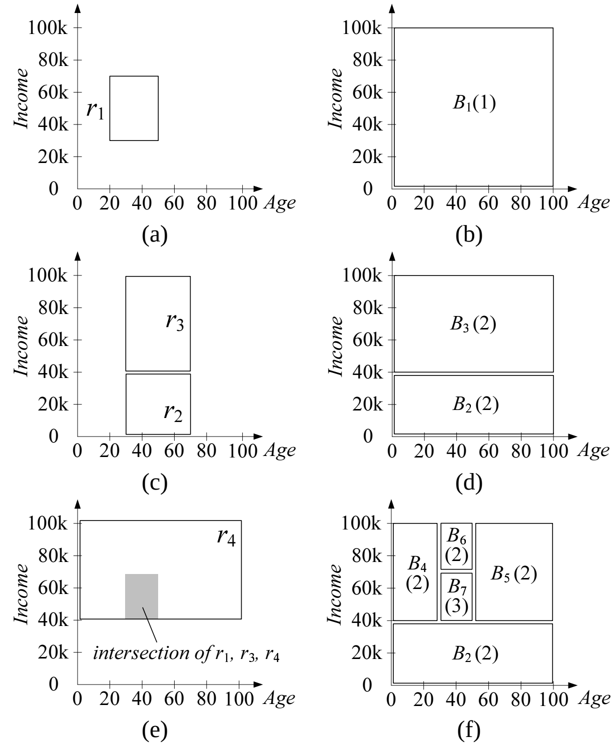


Figure 3: Illustration of our histogram approach

if $B.c \le \epsilon\lambda/2$ for every bucket $B \in \mathcal{H}$, we have $p(Q) \le \epsilon\lambda/2$ for any point $p \in \Omega$. Hence, by Lemma 2, $S_{L1}(Q) \le \epsilon\lambda$, indicating that $\mathcal{D}$ preserves ϵ-differential privacy (Theorem 1).

Query Processing. The above observation leads to the algorithm *Process* in Figure 2 for answering queries. Given a new query q with region r, *Process* identifies the set S_{buk} of buckets in $\mathcal{H}$ intersecting r. If any bucket in S_{buk} has a counter at least $\epsilon\lambda/2$, the answer ans for q is NULL, i.e., q is denied. Otherwise, *Process* reports $ans = q(\mathcal{D})$ (recall that $q(\mathcal{D})$ has included a Laplace noice), adds q to Q, and increases the counters of the buckets in S_{buk}.

When $|\mathcal{H}| < \theta$ (i.e., there is still space to store more buckets), counter increases may trigger bucket splits. Specifically, for any bucket $B \in S_{buk}$, in case $B.c = \epsilon\lambda/2$, *Process* invokes the *Split* sub-routine to decompose B in into new buckets. The details of *Split* will be elaborated shortly.

EXAMPLE 4. Suppose that the microdata table T is Table 1, and the maximum permissible popularity $\epsilon\lambda/2$ is 3. Initially, $\mathcal{H}$ has a single bucket B_1, which covers the entire data space, and its $B_1.c$ equals 0.

The first query to our statistical database $\mathcal{D}$ is the q_1 in Section 1.1), whose region r_1 is illustrated in Figure 3a. B_1 is the only bucket in $\mathcal{H}$ overlapping r_1 (i.e., $S_{buk} = \{B_1\}$ in the pseudocode of *Process*). Since $B_1.c = 0 \le \epsilon\lambda/2 = 3$, it is safe to answer q_1; hence, we report $q_1(\mathcal{D})$ to the user. Accordingly, Q becomes $\{q_1\}$, and $B_1.c$ equals 1. Figure 3b demonstrates the extent of B_1, and its counter $B_1.c$ in the bracket.

The next two queries to $\mathcal{D}$ are the q_2 and q_3 mentioned in Section 1.1, whose regions r_2 and r_3 respectively are depicted in Figure 3c. Both q_2 and q_3 are answerable, as can be verified in the same way as q_1. After returning $q_2(\mathcal{D})$ and $q_3(\mathcal{D})$, Q becomes $\{q_1, q_2, q_3\}$, and the counter of B_1 grows to 3, reaching the split threshold $\epsilon\lambda/2$. Thus, B_1 is decomposed (by the sub-routine *Split*) into B_2 and B_3, whose extents are shown in Figure 3d. The details

Algorithm *Split* (B)
```
/* B is a bucket to be decomposed */
U = the set of regions of the queries in Q that partially intersect B
if U ≠ ∅
   remove B from H
   r∩ = the intersection of all the regions in U
   if r∩ = ∅
      split B into buckets B' and B'' with the minimum B'.c + B''.c
      using the cutting lines passing the boundaries of the regions in U
   else
      repetitively split B by the cutting lines passing the boundaries of r∩
      until a bucket has extent r∩
   insert the new buckets into H with counters set to B.c
```

Figure 4: Bucket split algorithm

of the decomposition will become clear later. $B_2.c = 2$ because B_2 overlaps two queries q_1 and q_2 in Q. Likewise, $B_3.c$ is also 2.

The fourth query q_4 to $\mathcal{D}$ is:

q_4: SELECT COUNT(*) FROM T
WHERE *Age* $= *$ AND *Income* $\in$ [40000, 99999]

whose region r_4 is presented in Figure 3e. Among the two buckets B_2, B_3 in $\mathcal{H}$, only B_3 intersects r_4 (i.e., $S_{buk} = \{B_3\}$). Since $B_3.c = 2 < \epsilon\lambda/2$, q_4 is answerable; $\mathcal{D}$ returns $q_4(\mathcal{D})$, and updates Q to $\{q_1, q_2, q_3, q_4\}$. $B_3.c$ becomes 3, triggering a split. Decomposition of B_3 leads to 4 buckets B_4, B_5, B_6, and B_7, whose extents and counters are illustrated in Figure 3f. Finally, $\mathcal{H}$ includes totally five buckets.

It is worth mentioning that, since B_7 has a counter $3 = \epsilon\lambda/2$, all future queries whose regions intersect B_7 will be denied. $\square$

Bucket Decomposition. Figure 4 presents the details of *Split*, which *Process* deploys to decompose a bucket B whose counter $B.c$ equals $\epsilon\lambda/2$. *Split* begins by retrieving the set U of regions in R that *partially* intersect B (recall that R contains the regions of the set Q of previously-answered queries). If $U = \emptyset$, exactly $\epsilon\lambda/2$ regions in R fully contain B. In this case, splitting B does not lower its counter, because all points in B have popularity $\epsilon\lambda/2$ in Q. Hence, *Split* simply terminates, and keeps B in $\mathcal{H}$.

Next we focus on the case $U \neq \emptyset$. *Split* removes B from $\mathcal{H}$, computes the intersection $r_\cap$ of the regions in U. Then, it divides B using one or more *cuts*:

DEFINITION 6 (CUT). Let L be a $d-1$ dimensional plane in Ω that is perpendicular to an axis. The *cut* of B by L results in buckets B' and B'', which are separated by L, and their union is B. We say that L is a *cutting line*. $\square$

In case $r_\cap = \emptyset$, *Split* attempts all the cutting lines that go through a boundary of every region in U. Among those lines, *Split* decomposes B using the one that minimizes the sum of the counters of the new buckets, i.e., $B'.c + B''.c$ is the smallest. We aim to minimize $B'.c + B''.c$, because a smaller $B'.c$ ($B''.c$) allows us to answer more queries intersecting B' (B''), i.e., a lower value of $B'.c + B''.c$ leads to a larger number of admissible queries.

EXAMPLE 4 (CONTINUED). Let us revisit the moment in Example 4 when the counter $B_1.c$ reaches the split threshold $\epsilon\lambda/2$ = 3. At this point, $Q = \{Q_1, Q_2, Q_3\}$, and their regions r_1, r_2, r_3 are shown in Figures 3a and 3c. To decompose B_1, *Split* identifies $U = \{r_1, r_2, r_3\}$, since all these regions intersect B_1. Clearly, $r_\cap$ is empty (in fact, the intersection of r_2 and r_3 is already empty). Thus, *Split* tries to cut B_1 using the vertical/horizontal lines that contain the edges of r_1, ..., r_3. It can be verified that, among all those lines, the horizontal line *Income* = 40k is the best, achieving the smallest $B_1.c + B_2.c = 4$. $\square$

If $r_\cap \neq \emptyset$, B is decomposed into multiple buckets, one of which has the extent exactly $r_\cap$. *Split* accomplishes this using only the set S_{cut} of cutting lines that contain the boundaries of $r_\cap$. Clearly, S_{cut} has $2d$ lines. *Split* randomly picks one of them, and uses it to decompose B into B', B''. One of B', B'' is disjoint with $r_\cap$, and is retained in $\mathcal{H}$ directly. Suppose, without loss of generality, that B' is disjoint with $r_\cap$; then, B'' must cover $r_\cap$, and is split further using another cutting line from S_{cut}. This process is repeated, until the extent of a bucket is $r_\cap$. To understand why we decompose B in such a way, observe that any point in $r_\cap$ ($B - r_\cap$) is covered by exactly $\epsilon\lambda/2$ (at most $\epsilon\lambda/2 - 1$) queries. In other words, all queries that intersect $r_\cap$ should be denied, whereas any queries that cover only the points in $B - r_\cap$ can be answered. Therefore, we separate $r_\cap$ from the other points in B.

EXAMPLE 4 (CONTINUED). Consider the moment in Example 4 when the counter $B_3.c$ equals 3. At this time, $Q = \{Q_1, Q_2, Q_3, Q_4\}$, whose regions r_1, ..., r_4 can be found in Figures 3a, 3c, 3e. To decompose B_3, *Split* finds $U = \{r_1, r_3, r_4\}$. Note that r_2 is not included since it is disjoint with B_3. The intersection $r_\cap$ of all regions in U is the shaded area in Figure 3e. Therefore, S_{cut} has four cutting lines: *Age* = 30, *Age* = 50, *Income* = 40k, and *Income* = 70k, each of which contains an edge of $r_\cap$, respectively. B_3 is decomposed into B_4, B_5, B_6, B_7 by using the lines *Age* = 30, *Age* = 50, *Income* = 70k in this order. $\square$

Split can be implemented in $O(\epsilon\lambda \cdot \log(\epsilon\lambda))$ time. Since *Process* invokes *Split* at most (θ) times, it has a time complexity $O(\theta\epsilon\lambda \cdot \log(\epsilon\lambda))$, where θ is the number of buckets.

3.3 Limitation of Output Perturbation

We close this section with a theoretical result on the maximum number of queries that can be answered without violating ϵ-differential privacy. Given a query q, we define its *volume* as the percentage of points in Ω that qualify its WHERE condition. Specially, a query with predicate "$A_i = *$" on all A_i ($i \in [1, d]$) has a volume 1. Then:

LEMMA 3. *Consider any solution that (i) guarantees ϵ-differential privacy, and (ii) perturbs each query answer by a Laplace noise having magnitude λ. Let θ be the maximum number of queries that can be processed by such a solution. Then,*

1. *if each query has a fixed volume s ($0 < s < 1$),*
$$\theta < \frac{\epsilon\lambda}{2s(1-s)}; \tag{3}$$
2. *if each query has a volume at least s' and at most $1 - s'$ ($0 < s' \leq 1/2$),*
$$\theta < \frac{\epsilon\lambda}{s'}. \tag{4}$$

PROOF. Assume that the solution has answered a set Q of queries. Due to ϵ-differential privacy, by Theorem 1, $S_{L1}(Q) \leq \epsilon\lambda$. Let $n = |\Omega|$, and p_i ($1 \leq i \leq n$) be the i-th point in Ω. Without loss of generality, suppose that, among all points in Ω, p_1 has the largest popularity in Q. Then,

$$n \cdot p_1(Q) \geq \Sigma_{i=1}^{n} p_i(Q). \tag{5}$$

Let Q_1 be the set of queries in Q whose regions cover p_1, and $Q_2 = Q - Q_1$. So $p_1(Q) = |Q_1|$.

Consider the following proposition Z: $\forall i \in [2, n]$, $p_1(Q) - p_i(Q_1) + p_i(Q_2) \leq S_{L1}(Q)$. Assume for the moment that Z is valid (we will prove Z shortly). In the following, we will first show that, when each query in Q has a volume s, $|Q| < \frac{\epsilon\lambda}{2s(1-s)}$ holds.

Since each query in Q covers $n \cdot s$ points in Ω, we know

$$\Sigma_{i=1}^{n} p_i(Q) = |Q| \cdot n \cdot s, \tag{6}$$

which implies that $\Sigma_{i=2}^{n} p_i(Q_1) = |Q_1| \cdot (n \cdot s - 1)$, and $\Sigma_{i=2}^{n} p_i(Q_2) = (|Q| - |Q_1|) \cdot n \cdot s$. Furthermore, Equations 5 and 6 lead to $p_1(Q) \geq |Q| \cdot s$. Then, by proposition Z, we have $\Sigma_{i=2}^{n} S_{L1}(Q)$

$$\begin{aligned}
&\geq \Sigma_{i=2}^{n} p_1(Q) - \Sigma_{i=2}^{n} p_i(Q_1) + \Sigma_{i=2}^{n} p_i(Q_2) \\
&= |Q_1| \cdot (n-1) - |Q_1| \cdot (n \cdot s - 1) + (|Q| - |Q_1|) \cdot n \cdot s \\
&= p_1(Q) \cdot n \cdot (1 - 2s) + |Q| \cdot n \cdot s \\
&\geq |Q| \cdot s \cdot n \cdot (1 - 2s) + |Q| \cdot n \cdot s \\
&= 2|Q| \cdot (1 - s) \cdot n \cdot s.
\end{aligned}$$

This indicates $(n-1)S_{L1}(Q) \geq 2|Q| \cdot n(1-s)s$, leading to

$$|Q| \leq \frac{n-1}{n} \cdot \frac{S_{L1}(Q)}{2(1-s)s} < \frac{\epsilon\lambda}{2(1-s)s},$$

which establishes Equation 3.

Next, we will show that, when each query in Q has a volume in $[s', 1 - s']$, Equation 4 holds. Let $vol(q)$ denote the volume of any query q. By proposition Z, we know $\Sigma_{i=2}^{n} S_{L1}(Q)$

$$\begin{aligned}
&\geq \Sigma_{i=2}^{n} p_1(Q) - \Sigma_{i=2}^{n} p_i(Q_1) + \Sigma_{i=2}^{n} p_i(Q_2) \\
&= |Q_1| \cdot (n-1) - \sum_{q \in Q_1} (n \cdot vol(q) - 1) + \sum_{q \in Q_2} (n \cdot vol(q)) \\
&= |Q_1| \cdot n - \sum_{q \in Q_1} (n \cdot vol(q)) + \sum_{q \in Q_2} (n \cdot vol(q)) \\
&= n \cdot \sum_{q \in Q_1} (1 - vol(q)) + n \cdot \sum_{q \in Q_2} vol(q) \\
&\geq n \cdot \sum_{q \in Q_1} \big(1 - (1 - s')\big) + n \cdot \sum_{q \in Q_2} s' \\
&= n \cdot |Q| \cdot s'.
\end{aligned}$$

In other words, $(n-1)S_{L1}(Q) \geq |Q| \cdot n \cdot s'$, which implies that

$$|Q| \leq \frac{n-1}{n} \cdot \frac{S_{L1}(Q)}{s'} < \frac{\epsilon\lambda}{s'},$$

validating Equation 4.

It remains to prove proposition Z. Assume, on the contrary, that there exists $j \in [2, n]$ such that

$$p_1(Q) - p_j(Q_1) + p_j(Q_2) > S_{L1}(Q). \tag{7}$$

Let Q_3 be the set of queries in Q_1 whose regions contain p_j. Let t_1 and t_2 be the tuples corresponding to p_1 and p_j, respectively. Let T_1 (T_2) be a microdata table including only t_1 (t_2). By Definition 1, T_1 and T_2 are siblings. Since any query in Q_3 covers both p_1 and p_j, it holds that

$$\forall q \in Q_3,\ q(T_1) = q(T_2) = 1. \tag{8}$$

As each query in $Q_1 - Q_3$ covers p_1 but not p_j, we know

$$\forall q \in Q_1 - Q_3,\ q(T_1) = 1,\ q(T_2) = 0. \tag{9}$$

Furthermore,

$$\forall q \in Q_2,\ q(T_1) = 0,\ q(T_2) \geq 0. \tag{10}$$

Combining Equations 8, 9, and 10, $\Sigma_{q \in Q} |q(T_1) - q(T_2)|$

$$\begin{aligned}
&= \Sigma_{q \in Q_3} |q(T_1) - q(T_2)| + \Sigma_{q \in Q_1 - Q_3} |q(T_1) - q(T_2)| \\
&\quad + \Sigma_{q \in Q_2} |q(T_1) - q(T_2)| \\
&= \Sigma_{q \in Q_1} q(T_1) - \Sigma_{q \in Q_1} q(T_2) + \Sigma_{q \in Q_2} q(T_2).
\end{aligned} \tag{11}$$

As $\Sigma_{q \in Q_1} q(T_1) = p_1(Q)$, $\Sigma_{q \in Q_1} q(T_2) = p_j(Q_1)$, $\Sigma_{q \in Q_2} q(T_2) = p_j(Q_2)$, by Equations 7, 11, $\Sigma_{q \in Q} |q(T_1) - q(T_2)|$

$$\begin{aligned}
&= \Sigma_{q \in Q_1} q(T_1) - \Sigma_{q \in Q_1} q(T_2) + \Sigma_{q \in Q_2} q(T_2) \\
&= p_1(Q) - p_j(Q_1) + p_j(Q_2).
\end{aligned}$$

By our hypothetic assumption earlier, the above is greater than $S_{L1}(Q)$. Thus, we have arrived at $\Sigma_{q \in Q} |q(T_1) - q(T_2)| > S_{L1}(Q)$, which contradicts Definition 3. □

Equation 4 leads to the following corollary.

COROLLARY 2. *Let n be the total number of points in Ω. To guarantee ϵ-differential privacy, any solution, which perturbs each query answer by a Laplace noise with a magnitude λ, can process at most $n \cdot \epsilon\lambda$ queries with volumes in $(0, 1)$.*

PROOF. For any query with a volume in $(0, 1)$, it should contain at least 1 point (and at most $n - 1$ points) in Ω. Therefore, the volume of each query lies in $[1/n, 1 - 1/n]$. By Equation 4, the total number of allowable queries is at most $\epsilon\lambda/(1/n)$, which proves the corollary. □

Hence, any output-perturbation method leveraging Laplace noise can answer at most $O(n)$ queries (where $n = |\Omega|$), after which the statistical database will simply stop functioning. In contrast, the total number of possible queries on Ω is $\Theta(n^2)$. The next section presents a technique to remedy this drawback.

4. QUERY RELAXATION

The solution in Section 3 denies a query if answering it violates ϵ-differential privacy. As explained in Section 1.1, query denial reduces the utility of the database. In the sequel, we remedy the problem with query relaxation.

Specifically, let q^* be a query that is rejected by the statistical database $\mathcal{D}$. Query relaxation returns (i) the definition of a query $q^{*\prime}$, and (ii) a synthetic answer v for $q^{*\prime}$. In particular, $q^{*\prime}$ may not necessarily be the same as q^*, but in case they are not, $q^{*\prime}$ is similar to q^*. Furthermore, v is synthetic, because it derivation differs from the normal process that $\mathcal{D}$ uses to compute an answer (recall that, if $\mathcal{D}$ accepts a query, then the answer is obtained by adding a Laplace noise to the query's real result). In particular, v is synthesized by utilizing only the *reported* answers of the past queries. Remember that those queries and their reported answers are already publicly available. Thus, *query relaxation is using only the public knowledge to infer the result of q^**. This guarantees ϵ-differential privacy because, as mentioned in Section 1.1, whatever derived solely from public information remains public knowledge.

4.1 Compound

To avoid ambiguity, we say that $\mathcal{D}$ *accepts* a query if $\mathcal{D}$ returns a *perturbed answer* using the method in Section 3 (i.e., processing the query causes no privacy violation). Accepted queries are distinguished from the other *denied queries*, for which $\mathcal{D}$ produces *synthetic answers* via relaxation. Given any set S of accepted queries, its *total answer* equals the sum of the reported answers of all queries in S.

Let Q be the set of accepted queries in history, and q^* a denied query whose region is r^*. Relaxation looks for a subset P_+ of Q,

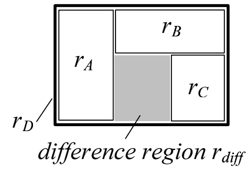


Figure 5: Illustration of a compound

containing accepted queries whose regions can be put together to form a rectangle similar to r^*. Then, we use the total answer of P_+ as the synthetic answer for q^*. However, some queries in P_+ may have overlapping regions in which case their intersections are over-counted. Therefore, to increase accuracy, relaxation also searches for another subset P_- of Q, involving queries whose regions correspond to the intersection areas in P_+. To cancel the effect of over-counting, we subtract the total answer of P_- from that of P_+. The pair of (P_+, P_-) constitutes a *compound*, which is formalized below:

DEFINITION 7 (COMPOUND). Two disjoint sets P_+ and P_- of queries constitute a *compound* P, if:

1. For each point p in the data space Ω, $p(P_+) - p(P_-)$ equals 0 or 1, where $p(P_+)$ and $p(P_-)$ are the popularities of p in P_+ and P_-, respectively.
2. All points $p \in \Omega$ satisfying $p(P_+) - p(P_-) = 1$ form a rectangle r_{diff}, which is the *difference region* of P. □

We refer to $|P_+ \cup P_-|$ as the *size* of P. As explained earlier, we compute a *synthetic answer* of P by

$$\sum_{q \in P_+} q(\mathcal{D}) - \sum_{q \in P_-} q(\mathcal{D}). \tag{12}$$

where $q(\mathcal{D})$ is the reported answer of an accepted query q. Intuitively, condition 1 of Definition 7 requires no over-counting at all in the synthetic answer of P. The difference region r_{diff}, formulated in condition 2, is exactly the region that the synthetic answer corresponds to. Furthermore, r_{diff} is also the relaxed query $q^{*\prime}$ returned to the user. Hence, condition 2 demands r_{diff} to be a rectangle.

EXAMPLE 5. Assume that Q consists of four queries q_A, q_B, ..., q_D, whose regions r_A, ..., r_D are illustrated in Figure 5. Let $P_+ = \{q_D\}$ and $P_- = \{q_A, q_B, q_C\}$. Then, $P = (P_+, P_-)$ is a compound. Specifically, for any point p outside r_D, its popularities $p(P_+)$ and $p(P_-)$ in P_+ and P_- respectively are both 0. For any point p inside r_D but outside the grey area, $p(P_+)$ and $p(P_-)$ are both 1. For any point p in the grey area, $p(P_+)$ = 1 whereas $p(P_-)$ = 0. Hence, condition 1 is fulfilled. Furthermore, condition 2 is also satisfied because $p(P_+) - p(P_+) = 1$ only when p is in the shaded area, which is thus the difference region of P. The synthetic answer of P equals $q_D(\mathcal{D}) - (q_A(\mathcal{D}) + q_B(\mathcal{D}) + q_C(\mathcal{D}))$. □

Ideally, the difference region r_{diff} of a compound P should be identical to the region r^* of the denied query q^*. When this is not true, we need a metric for quantifying the quality of a compound. The next subsection addresses this issue.

4.2 Relaxation Error

Let r be an axis-parallel rectangle in the space Ω. Denote its projection on the i-th dimension ($1 \le i \le d$) as $[r.x_i, r.y_i]$. Also, use $A_i.max$ ($A_i.min$) to represent the maximum (minimum) value on the i-th axis. As mentioned earlier, given a denied query q^* with region r^*, we want to find a compound P whose difference region r_{diff} is as similar to r^* as possible. To measure the similarity between r_{diff} and r^*, we introduce the following metric:

Algorithm *Patch-check* (P, q)
/* P is a compound and q an accepted query */
1. r_{diff} = the difference region of P
2. r = the region of q^*
3. if $r_{diff} \cap r = \emptyset$
4.   if the union of r_{diff} and r is a rectangle
5.   if the relaxation error drops after including q in P_+
6.   return P_+
7. else if r_{diff} covers r and $r_{diff} - r$ is a rectangle
8.   if the relaxation error drops after including q in P_-
9.   return P_-
10. return NULL

Figure 6: Checking whether a query is a patch

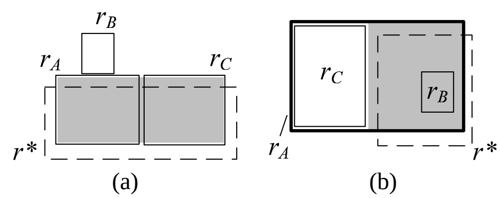


Figure 7: Illustration of *Patch-check*

DEFINITION 8 (RELAXATION ERROR). Let P be a compound and q^* a denied query with region r^*. The *relaxation error* $E(P, q)$ equals

$$\frac{1}{d} \sum_{i=1}^{d} \left(w_i \cdot \frac{\left|r^*.x_i - r_{diff}.x_i\right| + \left|r^*.y_i - r_{diff}.y_i\right|}{A_i.max - A_i.min} \right) \tag{13}$$

where weights w_1, ..., w_d can be any positive values. □

Weight w_i ($i \in [1, d]$) is a constant reflecting the importance of dimension A_i. A large w_i means that A_i is imperative, such that even a small difference between r^* and r_{diff} along this dimension may cause heavy penalty. A small w_i achieves the opposite effect. For simplicity, in the sequel, we assume $w_1 = ... = w_d = 1$ because our solutions extend to arbitrary weights directly.

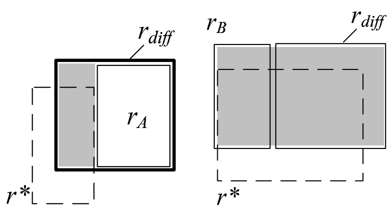
Given a compound P, Equation 13 suggests an easy way to identify which query can be inserted in P to reduce relaxation error. We refer to such a query as a *patch*:

DEFINITION 9 (PATCH). Let Q the set of accepted queries and P = $\{P_+, P_-\}$ be a compound. Consider a query $q \in Q$ that does not belong to P yet. We say that q is a *positive* (*negative*) *patch* if, after including q in P_+ (P_-), (i) P remains a compound and (ii) $E(P, q^*)$ decreases. □

Figure 6 gives an algorithm *Patch-check* for verifying whether a query q is a patch for a compound P = $\{P_+, P_-\}$. In case it is, *Patch-check* indicates whether q should be added to P_+ or P_-. If q is not a patch, the algorithm returns NULL. Next, we illustrate the algorithm using an example.

EXAMPLE 6. Assume that Q contains q_A, q_B, q_C whose regions r_A, ..., r_C are shown in Figure 7a. Rectangle r^* is the region of a denied query q^*. Consider a compound $P = (P_+, P_-)$, where $P_+ = \{r_A\}$ and $P_- = \emptyset$. The difference region r_{diff} of P is r_A.

To see whether q_B is a patch, *Patch-check* starts by noticing that r_B is disjoint with r_{diff} (Line 3 of Figure 6). In this case, the algorithm examines if the union of r_{diff} and r_B is a rectangle (Line 4). The answer is negative, and therefore, *Patch-check* returns NULL. The region r_C of q_C, on the other hand, is disjoint with r_{diff}, and meanwhile, can union r_{diff} into a rectangle. Hence, *Patch-check* examines whether inclusion of q_C in P_+ reduces the relaxation error (Line 5). For this purpose, it obtains the new r_{diff} (if r_C is indeed inserted in P_+), which is the shaded area in Figure 7a. Clearly,



(a) (b)

Figure 8: Artificial queries reduce relaxation error

compared to the original r_{diff}, the shaded area is more similar to r^*, implying lower relaxation error. Therefore, q_C is a positive patch, and *Patch-check* returns P_+ (Line 6).

Consider another example, where r_A, r_B, r_C, r^* are demonstrated in Figure 7b (r_A is the bold rectangle). Again, suppose $P = (P_+, P_-)$, where $P_+ = \{r_A\}$ and $P_- = \emptyset$, and apparently, the r_{diff} of P is r_A. Let us apply *Patch-check* to verify whether q_B is a patch. Since r_B intersects r_{diff}, *Patch-check* executes Line 7, and proceeds only if (i) r_{diff} encloses r_B and (ii) the difference between r_{diff} and r_B is a rectangle. Here, although (i) is true, (ii) is not. Hence, *Patch-check* finishes with NULL. On the other hand, r_C satisfies both (i) and (ii), and thus, *Patch-check* proceeds to inspect the relaxation error after adding q_C to P_- (Line 8). The shaded area in Figure 7 shows the new r_{diff} (if q_C is in P_-), which is a better approximation of r^* than the original r_{diff}. Hence, q_C is a negative patch, and the algorithm terminates with P_- (Line 9). $\square$

4.3 Artificial Patches

So far we have assumed that a compound P contains only the queries in Q that are *explicitly* issued by users in the past. This section explores another possibility: we can also dynamically generate a query, force the database to process it *normally* (i.e., using the solution in Section 3), and then, use its perturbed answer to obtain a better synthetic answer for the denied query q^*.

To illustrate, consider Figure 8a, where r^* is the region of q^*, and r_{diff} (the bold rectangle) is the difference region of the current compound P. Obviously, r_{diff} is a poor approximation of r^*. Imagine, however, that we *had* an accepted query q_A in Q whose region is r_A. This query is a negative patch, because its inclusion in P_- shrinks r_{diff} to the shaded area, which is significantly more similar to r^*. In fact, even though q_A is not in Q, we can instruct the database $\mathcal{D}$ to process it (as an accepted query) *right away*, after which q_A can be incorporated in Q, and hence, becomes a candidate patch to be selected by *Patch-check* (Figure 6).

In Figure 8a, the artificial query q_A aligns with the right edge of r^*. Sometimes, it is better to align with the left edge of r^*. For example, let us examine Figure 8b, where r_B is the region of an artificial query q_B. Apparently, q_B a positive patch, as its insertion in P_+ expands r_{diff} to the shaded area, which has much lower relaxation error. Similarly, artificial queries may also be created on the y-dimension, by aligning with the upper and lower edges of r^*, respectively.

In general, given a compound P with difference region r_{diff}, we prepare an *artificial patch-set* S_{arti} as follows. First, S_{arti} is initiated with $2d$ artificial queries, each of which aligns with a boundary of r_{diff} (details clarified shortly). Then, we invoke *Patch-check* to eliminate those queries in S_{arti} that are not patches (i.e., they do not reduce the relaxation error). Some remaining queries may be denied by $\mathcal{D}$ due to ϵ-differential privacy (i.e., if they intersect a bucket in the histogram $\mathcal{H}$ with counter $\epsilon\lambda/2$; see Section 3), and are also removed from S_{arti}. The resulting S_{arti} is the final artificial patch-set.

It remains to explain how to obtain the initial $2d$ queries in S_{arti}.

Algorithm *Relax* (q^*, ξ)
/* q^* is a denied query, and ξ the maximum compound size */
1. q = the query in Q minimizing $E(P, q^*)$, where $P_+ = \{q\}$, $P_- = \emptyset$, and $P = \{P_+, P_-\}$
2. ans = the reported answer of q
3. while the size of P is smaller than ξ
4. M = the set of queries in Q that are patches for P /* using *Patch-check* in Figure 6 */
5. $M = M \cup S_{arti}$ /* See Section 4.3 about deriving S_{arti} */
6. if $M = \emptyset$ then goto Line 14
7. else
8. $q =$ the patch in M whose insertion in P minimizes $E(P, q)$
9. if $q \notin Q$ then $x = \mathit{Process}(q)$
10. else x = the reported answer of q
11. if q is a positive patch
12. $P_+ = P_+ \cup \{q'\}$; $ans = ans + v$
13. else $P_- = P_- \cup \{q'\}$; $ans = ans - v$
14. return ans and the difference region of P

Figure 9: The relaxation algorithm

Specifically, the $(2i-1)$-th $(1 \le i \le d)$ query has a region whose projection on dimension A_j is:

$$\begin{cases} [r_{diff}.x_j, r_{diff}.y_j] & \text{if } j \neq i \\ [r_{diff}.x_i, r^*.x_i) & \text{if } j = i \text{ and } r^*.x_i > r_{diff}.x_i \\ [r^*.x_i, r_{diff}.x_i) & \text{otherwise} \end{cases}$$

Similarly, the region of the $2i$-th query has the following projection on A_j:

$$\begin{cases} [r_{diff}.x_j, r_{diff}.y_j] & \text{if } j \neq i \\ (r_{diff}.y_i, r^*.y_i] & \text{if } j = i \text{ and } r^*.y_i > r_{diff}.y_i \\ (r^*.y_i, r_{diff}.y_i] & \text{otherwise} \end{cases}$$

4.4 Probabilistic Accuracy

Recall that, given a compound P, we return a synthetic answer v calculated by Equation 12, and a relaxed query $q^{*\prime}$. The value v is actually an unbiased estimate the real result $q^{*\prime}(T)$, but has a variance proportional to the size of P:

LEMMA 4. *Equation 12 has the expected value $q^{*\prime}(T)$, and its variance is $2\lambda^2 \cdot |P_+ \cup P_-|$, where λ is the noise magnitude of $\mathcal{D}$.*

PROOF. For any query q in P_+ or P_-, let δ_q be the noise that $\mathcal{D}$ injects into $q(\mathcal{D})$. Denote v as the value of Equation 12.

$$\begin{aligned} v &= \Sigma_{q \in P_+} q(\mathcal{D}) - \Sigma_{q \in P_-} q(\mathcal{D}) \\ &= \Sigma_{q \in P_+} \big(q(T) + \delta_q\big) - \Sigma_{q \in P_-} \big(q(T) + \delta_q\big) \\ &= \Sigma_{q \in P_+} q(T) - \Sigma_{q \in P_-} q(T) + \Sigma_{q \in P_+} \delta_q - \Sigma_{q \in P_-} \delta_q. \end{aligned}$$

By Equation 1, the mean and variance of $\Sigma_{q \in P_+} \delta_q - \Sigma_{q \in P_-} \delta_q$ are 0 and $2\lambda^2 \cdot |P_+ \cup P_-|$, respectively. Hence, v has an expected value $\Sigma_{q \in P_+} q(T) - \Sigma_{q \in P_-} q(T)$, and variance $2\lambda^2 \cdot |P_+ \cup P_-|$. Next, we will show that $\Sigma_{q \in P_+} q(T) - \Sigma_{q \in P_-} q(T) = q^{*\prime}(T)$.

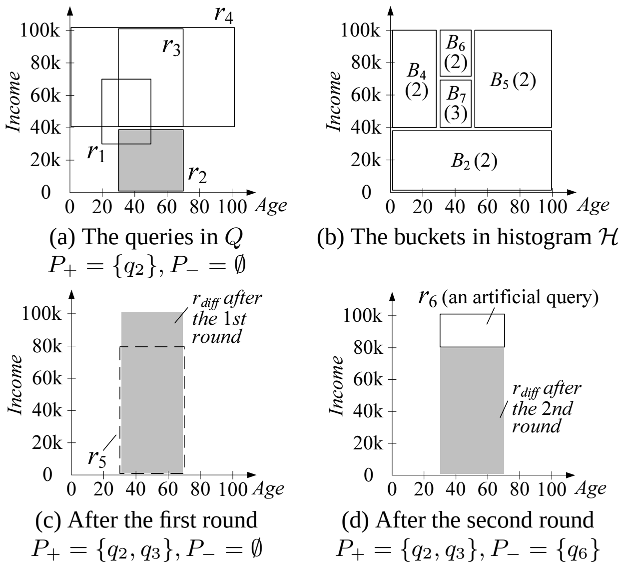
Consider the i-th $(1 \le i \le |T|)$ tuple t_i in T. Let p_i be the point representation of t_i in Ω, and $G = \{p_i \mid 1 \le i \le |T|\}$. Thus,

$$\Sigma_{q \in P_+} q(T) = \Sigma_{p \in G} p(P_+), \text{ and } \Sigma_{q \in P_-} q(T) = \Sigma_{p \in G} p(P_-).$$

Let r_{diff} be the difference region of P, which is also the region $r^{*\prime}$ of $q^{*\prime}$. By Definition 7, for any point $p \in \Omega$, $p(P_+) - p(P_-) = 1$ if $p \in r_{diff}$; otherwise $p(P_+) - p(P_-) = 0$. Hence, $\Sigma_{q \in P_+} q(T) - \Sigma_{q \in P_-} q(T) = \Sigma_{p \in G} \big(p(P_+) - p(P_-)\big)$

$$\begin{aligned} &= \Sigma_{p \in G \cap r_{diff}} \big(p(P_+) - p(P_-)\big) + \Sigma_{p \in G - r_{diff}} \big(p(P_+) - p(P_-)\big) \\ &= \Sigma_{p \in G \cap r_{diff}} 1 + \Sigma_{p \in G - r_{diff}} 0 \quad = \quad \Sigma_{p \in G \cap r^{*\prime}} 1 \\ &= \Sigma_{p \in G} p(\{q^{*\prime}\}) \quad = \quad q^{*\prime}(T). \end{aligned}$$

which completes the proof. $\square$



(a) The queries in Q $P_+ = \{q_2\}, P_- = \emptyset$

(b) The buckets in histogram $\mathcal{H}$

(c) After the first round $P_+ = \{q_2, q_3\}, P_- = \emptyset$

(d) After the second round $P_+ = \{q_2, q_3\}, P_- = \{q_6\}$

Figure 10: Illustration of *Relax*

Since the variance of the synthetic answer grows with the size of P, we allow the user to specify an upper-bound ξ on the size of a compound, i.e., there can be at most ξ queries in $P_+ \cup P_-$. The value of ξ controls the tradeoff between relaxation error and query accuracy: a larger ξ leads to compounds consisting of more queries, which lowers relaxation error but increases the noise in the query results; on the other hand, a smaller ξ ensures less noisy query answers, but may incur higher relaxation error.

4.5 Relaxation Algorithm

Based on the previous analysis, Figure 9 formally presents the query relaxation algorithm *Relax*. Given a denied query q^*, *Relax* starts with a simple compound P whose P_- is empty, and its P_+ contains the query in Q (the set of accepted queries) most similar to q^*. Then, *Relax* proceeds in rounds, each of which adds a query to P to minimize the relaxation error. Such a query is chosen from both Q and the artificial patch set S_{arti} computed as in Section 4.3. More rounds are carried out until either the size of P has reached the upper bound ξ, or no more patch can be found.

EXAMPLE 7. Assume that $\mathcal{D}$ has accepted the set Q of queries q_1, q_2, q_3, q_4 before, whose regions r_1, ..., r_4 are illustrated in Figure 10a. At this point, the histogram $\mathcal{H}$ has the buckets in Figure 10b, and the largest permissible bucket counter $\epsilon\lambda/2$ equals 3 (for ensuring ϵ-differential privacy). Now, $\mathcal{D}$ receives a new query q_5 whose region r_5 is shown in Figure 10c. $\mathcal{D}$ denies q_5, because r_5 intersects a bucket B_7 , whose counter 3 equals $\epsilon\lambda/2$. Then, $\mathcal{D}$ invokes *Relax* to derive a synthetic answer. Assume the maximum compound size ξ to be 3.

Among all the queries in Q, q_2 is the most similar to q_5; hence, *Relax* initializes $P_+ = \{q_2\}$ and $P_- = \emptyset$. Clearly, the difference region r_{diff} of P is r_2, i.e., the shaded area in Figure 10a.

The algorithm enters the first round. *Relax* builds a set M of patches of P. For this purpose, it employs *Patch-check* (Figure 6) to examine every query in Q that is not in P yet. The examination reveals that q_3 is a positive patch; hence, $M = \{q_3\}$. Then, *Relax* computes the artificial patch-set S_{arti} in the way described in Section 4.3, and adds all queries of S_{arti} to M. It can be verified that here $S_{arti} = \emptyset$, thus causing no change in M. As q_3 is the only element in M, it is inserted in P_+ (remember that q_3 is a positive patch), which thus becomes $\{q_2, q_3\}$. This changes the difference region r_{diff} to be the shaded area in Figure 10c.

Parameter	Values
noise magnitude λ	$\mathbf{2000}$
histogram size threshold θ	10^2, 10^3, 10^4, $\mathbf{10^5}$, 10^6
ϵ	0.1, 0.2, **0.3**, 0.4, 0.5
query volume s	1%, 2%, **4%**, 8%, 16%
compound size threshold ξ	1, 2, **3**, 4, 5

Table 2: Parameters and examined values

In the second round, *Relax* creates a set M of patches in the same manner. This time, no query from Q is added to M. The artificial patch-set S_{arti}, on the other hand, has a negative patch q_6, whose region r_6 is given in Figure 10d. Thus, M includes only q_6, which is placed in P_-. As a result, the difference region r_{diff} shrinks to the shaded area of Figure 10d. At this time, $P_+ = \{q_2, q_3\}$ and $P_- = \{q_6\}$.

Now that the size of P has reached the upper bound $\xi = 3$, *Split* finishes, and returns the synthetic answer of P, and the final r_{diff}. After this, q_6 needs to included in Q (which is now $Q = \{q_1, q_2, q_3, q_4, q_6\}$) because, as explained in Section 4.3, an artificial query is processed normally using the solution in Section 3. □

Each round of *Relax* examines the queries in Q once, which takes $O(\theta\epsilon\lambda)$ time because Q contains at most $\theta\epsilon\lambda/2$ queries, where θ is the number of buckets in the dynamic histogram. Since there are at most ξ rounds, *Relax* runs in $O(\xi\theta\epsilon\lambda)$ time.

5. EXPERIMENTS

This section experimentally evaluates the effectiveness of the proposed solutions. We use a real dataset CENSUS (obtainable from *http://www.ipums.org*) with one million tuples, each storing the information of an American. It has four attributes: *Age*, *Education*, *Occupation*, and *Income*, whose domain sizes are 79, 14, 23, and 100, respectively. We aim at guaranteeing ϵ-differential privacy with a noise magnitude $\lambda = 2000$. This choice of λ ensures that the expected absolute error of each query answer is a small value 2000 (as explained in Section 2), which accounts for only 0.2% of the cardinality of CENSUS.

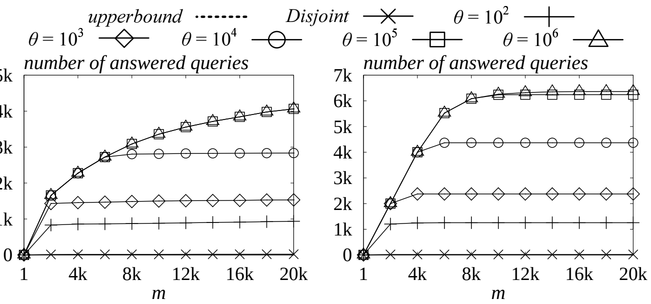
Each query has the form: *select count(*) from CENSUS where* $A_1 \in [x_1, y_1]$ *and* $A_2 \in [x_2, y_2]$. Here, A_1 and A_2 are two random attributes of CENSUS. Interval $[x_i, y_i]$ falls in the domain of A_i ($1 \le i \le 2$), and its length $y_i - x_i$ equals $\sqrt{s} \cdot (A_i.max - A_i.min)$, where $A_i.max$ ($A_i.min$) is the maximum (minimum) value in the domain of A_i, and s the query volume (defined in Section 3.3). The center z_i of $[x_i, y_i]$ follows one of the following distributions, which reflect the patterns of users' queries in practice [7]:

- *Data*: $z_i = t[A_i]$, where t is a tuple randomly selected from CENSUS.
- *Uniform*: z_i is a random value in the domain of A_i.

A (*Data-* or *Uniform-*) workload contains 20k queries with an identical s obeying the same distribution.

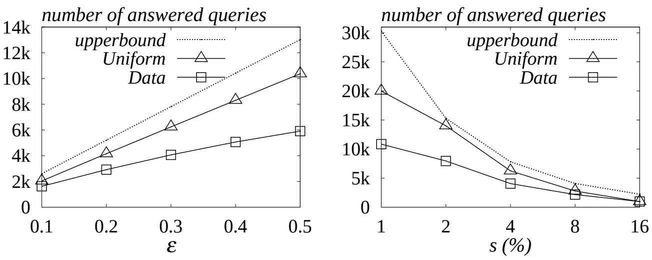
Table 2 summarizes the parameters examined in our experiments. Unless otherwise stated, each parameter is set to its default value (bold in the table) in each experiment. All the experiments are accomplished on a computer with a 3 GHz Pentium IV CPU and one gigabytes memory.

Processing Capacity without Relaxation. The first set of experiments studies the number of queries that can be answered by our *Histogram* approach (Section 3) without query relaxation. For comparison, we implement the only existing solution [13] that ensures ϵ-differential privacy in handling count queries. This solution,



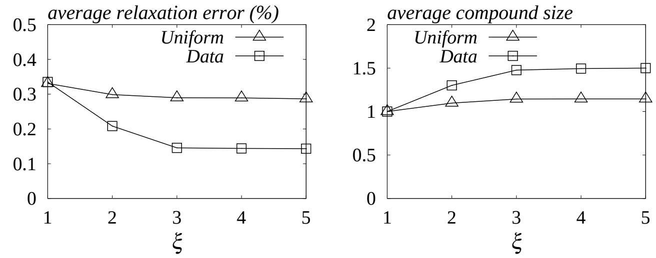
(a) *Data*-workload (b) *Uniform*-workload

Figure 11: Progressive processing behavior



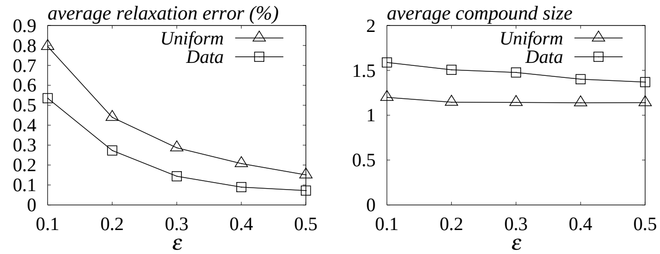
(a) Vs. ϵ (b) Vs. s

Figure 12: Num. of answered queries vs. ϵ, s



(a) Average relaxation error (b) Average compound size

Figure 13: Relaxation quality vs. ξ



(a) Average relaxation error (b) Average compound size

Figure 14: Relaxation quality vs. ϵ

referred to as *Disjoint*, processes an incoming query, if and only if its region does not overlap any of the queries answered previously.

In the experiment of Figure 11a, we submit the queries in a *Data*-workload to the underlying statistical database, and measure the number of processed queries, as a function of the number submitted. The figure demonstrates the results of *Disjoint*, *Histogram* adopting various numbers θ of buckets, and the theoretical upper bounds given by Lemma 3. Figure 11b illustrates the results of a similar experiment with a *Uniform*-workload. For each θ, the curve of *Histogram* initially increases because, during this period, the bucket counters are smaller than the limit $\epsilon\lambda/2$, thus permitting additional queries to be processed. The curve eventually turns horizontal, when the counters have reached the limit.

We use the term *processing capacity* to refer to the total number of queries in a workload that are answered by the database. Observe that the capacity of *Histogram* grows along with θ. This is because a histogram with more buckets provides a better estimate of $C(Q)$, and hence, reduces the chance of denying a query that could have been processed (if the real $C(Q)$ was maintained). Nevertheless, we witness no obvious gain by raising θ beyond 10^5, implying that $\theta = 10^5$ already offers adequate precision for maximizing the processing capacity. When θ is fixed, *Histogram* is able to answer more queries in a *Uniform*-workload than in a *Data*-workload. This is due to the fact that, uniform queries have less overlap in their regions, which leads to a lower $C(Q)$, and hence, fewer query denials.

For uniform queries and $\theta = 10^5$, the processing capacity of *Histogram* approaches the upper bound, which confirms the effectiveness of the proposed bucket maintenance algorithm. Since an upper bound assumes an "ideal" query distribution, it is reasonable for the actual capacity to be lower, especially given a "bad" distribution such as *Data*. Notice that *Histogram* has significantly higher capacity than *Disjoint*. Since this is true in all the subsequent experiments, we omit *Disjoint* in the following diagrams.

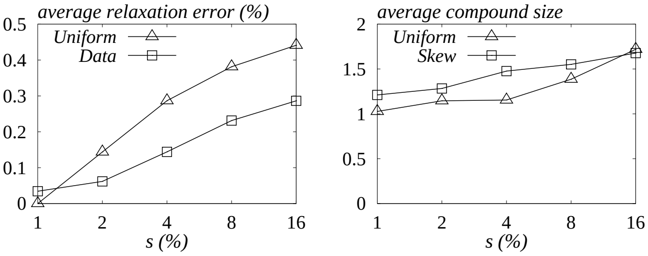
Next, we investigate the effects of ϵ and s on the processing capacity of *Histogram*. Figure 12a (12b) plots the actual capacity as a function of ϵ (s) for workloads of both distributions, together with the upper bounds. The capacity increases linearly with ϵ. This is expected, because the capacity is proportional to the limit $\epsilon\lambda/2$ on $C(Q)$, which, in turn, is linear to ϵ. On the other hand, a greater s results in a smaller capacity, since handling queries with larger regions causes faster growth of $C(Q)$.

Quality of Relaxation. The effectiveness of query relaxation (Section 4) is determined by: (i) the relaxation error (calculated by Equation 13) and (ii) the size of the final compound. The former indicates the amount of modification to the original query's predicates, whereas the latter determines the variance of a synthetic answer (see Lemma 4).

By varying ξ from 1 to 5, Figure 13a (13b) illustrates the average relaxation errors (compound sizes) of the queries that demand relaxation in *Data*- and *Uniform*-workloads, respectively. The average error is very small, indicating that a compound region used to derive a synthetic answer is almost identical to the original query region. The error decreases as ξ escalates, since allowing a larger compound raises the chance of finding a good compound (whose region incurs little relaxation error). The average compound size is fairly low, implying a small variance in the reported answers. Note that a compound size can be well below ξ, because the relaxation algorithm may terminate before the size reaches ξ.

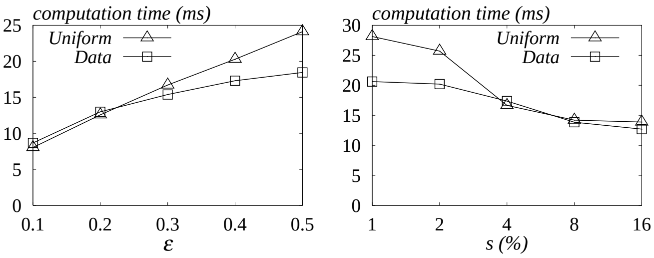
In Figure 14a (14b), we plot the average relaxation error (compound size) as a function of ϵ, when this parameter distributes from 0.1 to 0.5. Both factors decrease as ϵ becomes larger. To understand this, recall that a greater ϵ allows the database to process more queries (see Figure 12a), rendering a larger set Q usable by relaxation, and thus, enhancing relaxation quality. Figures 15a and 15b demonstrate the relaxation error and compound size, as s is varied between 1% and 16%. The two factors increase with s, which can again be explained by the relationship between the relaxation quality and the database's processing capacity (c.f. Figure 12b). In all cases, the relaxation error and compound size remain at very low levels, confirming the usefulness of our synthetic answers.

Computation Overhead. In the next set of experiments, we evaluate the average processing time required by our technique in answering queries. Figures 16a and 16b plot the computation overhead as a function of ϵ and s, respectively. The overhead escalates

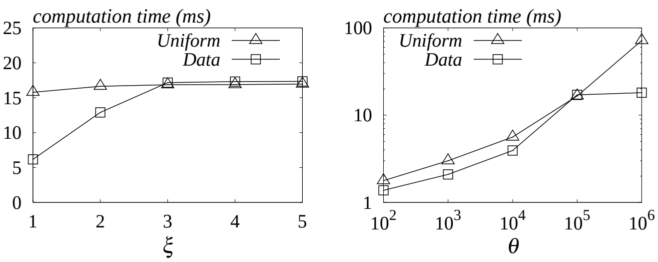


(a) Average relaxation error (b) Average compound size

Figure 15: Relaxation quality vs. s



(a) Vs. ϵ (b) Vs. s



(c) Vs. ξ (d) Vs. θ

Figure 16: Computation time

with the increase of ϵ (decrease of s), due to the following reasons. First, a larger ϵ leads to a greater processing capacity, as shown in Figure 12a. In turn, a high processing capacity renders the maintenance of the dynamic histogram less efficient, because each execution of *Split* requires a scan through all previously answered queries (see Line 1 in Figure 2). Consequently, the computation time increases with ϵ. On the other hand, a larger s results in a smaller processing capacity (see Figure 12b), and hence, a lower computation cost.

Figure 16c demonstrates the computation overhead as a function of ξ. The overhead increases with ξ, since a greater ξ enables our technique to utilize larger compounds (for query relaxation), which, however, require more time to construct. In Figure 16d, we plot the processing overhead, varying θ from 10^2 to 10^6. The overhead escalates with the increase of θ. This is because, a larger θ allows more buckets in the dynamic histogram, which entails higher processing cost, since our technique needs to inspect all histogram buckets to decide whether a query is answerable. Interestingly, when $\theta = 10^6$, the query overhead of *Data*-workload is much lower than that of *Uniform*-workload. To understand this, observe that the number of histogram buckets increases, only when the statistical database processes an answerable query (see Figure 2). Since *Data*-workload permits a smaller processing capacity than *Uniform*-workload, few histogram buckets are created for *Data*-workload, and thus, the computation overhead is lower. This phenomenon does not occur when $\theta \leq 10^5$, because the maximum numbers of histogram buckets entailed by each workloads is larger than 10^5, i.e., given $\theta \leq 10^5$, our technique have to utilize all θ buckets to process each workload, and hence, the computation overhead for both workloads is similar.

6. RELATED WORK

Output perturbation is first studied by the statistics community (see [2] for a survey). In particular, Denning [10] devises a method that proposes to answer queries on a random sample set of the underlying data; Fellegi and Phillips [15] devises a method that rounds each query result to the nearest multiple of a pre-defined number, while Achugbue and Chin [1] and Dalenius [9] investigate variations of this method. As pointed out in [12], however, the existing approaches in the statistics literature mainly address the utility of perturbed query results, without providing solid guarantees on privacy preservation, which severely limits their practicability.

In [11], Dinur and Nissim provide the first formal study on the amount of noise needed by any output perturbation scheme to ensure privacy in count queries. They show that, if an unlimited number of queries are allowed, the noise in each query answer must be linear to the dataset cardinality n; otherwise, an adversary may be able to restore the entire dataset precisely from the query results. As an unfortunate implication, when the dataset is sizable, query answers will have to be erroneous to avoid privacy disclosure. Dwork et al. [14] further prove that, even if the statistical database employs arbitrary noise in answering 0.269 fraction of the queries, and returns relatively accurate answers for the rest, an adversary can still reconstruct most tuples in the dataset.

To circumvent the problem, Blum et al. [6] propose a solution that permits only $o(n)$ count queries, but provides more accurate answers. This solution is subsumed by the differential privacy mechanism [13], which allows a larger number of queries and offers a higher degree of privacy protection. McSherry and Talwar [21] extend differential privacy for arbitrary queries, while Nissim et al. [23] improve the techniques in [13] by taking into account the *smooth sensitivity* of the queries.

Besides output perturbation, *query restriction* and *input perturbation* are also popular techniques for implementing statistical databases. Specifically, query restriction [8, 18, 22] works by denying queries that may lead to privacy breach, and returning exact answer for the other queries. Compared to output perturbation, this technique offers more useful query results, but weaker privacy protection. In particular, none of the existing query restriction technique can achieve ϵ-differential privacy.

When input perturbation is adopted, the statistical database first sanitizes the microdata with *generalization* [25, 26] or *random perturbation* [3, 4], and then processes queries using the sanitized data. The major advantage of input perturbation is that it is able to answer any number of queries. Nevertheless, the benefit is at the cost of sacrificing query accuracy. Dwork et al. [13] prove that, for practical datasets, random perturbation necessarily incurs larger error than output perturbation, in achieving ϵ-differential privacy. They also show that generalization cannot be used to ensure ϵ-differential privacy at all.

7. CONCLUSIONS

Although ϵ-differential privacy has been established as an important paradigm for statistical databases, it remains unclear whether the paradigm can be efficiently applied when the incoming (count) queries have arbitrary predicates. This paper provides a pessimistic answer, by proving that evaluating ϵ-differential privacy is NP-hard. Fortunately, as the second step, we show that it is possible to efficiently enforce this paradigm in a conservative manner. Our results lead to a histogram approach, which enables the processing of a majority of queries that qualify ϵ-differential privacy. Furthermore, given a query that violates the paradigm, our relaxation technique still provides a useful answer, as opposed to simply denying the query completely as in previous solutions.

Our work also opens several avenues for future research. First, in this paper we concentrate on statistical databases that answer count queries. It is interesting to investigate whether our solutions can be adapted to support other aggregate queries (e.g., SUM, MIN, MAX) as well. Second, the proposed solutions assume that there are no updates in the microdata. We plan to study extensions for the scenarios where only insertions are possible (i.e., append-only), and both insertions and deletions are allowed. Finally, our method is designed for relational tables. It is a challenging problem to devise output perturbation techniques for other types of microdata such as social networks, locations of moving objects, strings, etc.

Acknowledgements

This work was supported by CERG grants CUHK 4161/07 and CUHK 1202/06 from the RGC of HKSAR. The authors would like to thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] J. O. Achugbue and F. Y. L. Chin. The effectiveness of output modification by rounding for protection of statistical data bases. *Canadian Journal of Operational Research and Information Processing*, 17:208–218, 1979.

[2] N. R. Adam and J. C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Computing Surveys*, 21(4):515–556, 1989.

[3] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. of ACM Management of Data (SIGMOD)*, pages 439–450, 2000.

[4] R. Agrawal, R. Srikant, and D. Thomas. Privacy preserving olap. In *Proc. of ACM Management of Data (SIGMOD)*, pages 251–262, 2005.

[5] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 273–282, 2007.

[6] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the sulq framework. In *PODS*, pages 128–138, 2005.

[7] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *Proc. of ACM Management of Data (SIGMOD)*, pages 211–222, 2001.

[8] F. Y. L. Chin. Security problems on inference control for sum, max, and min queries. *J. ACM*, 33(3):451–464, 1986.

[9] T. Dalenius. A simple procedure for controlled rounding. *Statistik Tidskrift*, 3:202–208, 1981.

[10] D. E. Denning. Secure statistical databases with random sample queries. *ACM Trans. Database Syst.*, 5(3):291–315, 1980.

[11] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *PODS*, pages 202–210, 2003.

[12] C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[14] C. Dwork, F. McSherry, and K. Talwar. The price of privacy and the limits of lp decoding. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 85–94, 2007.

[15] P. Fellegi and J. L. Phillips. Statistical confidentiality: Some theory and applications to data dissemination. *Annals of Economic and Social Measurement.*, 3(2):399–409, 1974.

[16] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1:237–267, 1976.

[17] V. Iyengar. Transforming data to satisfy privacy constraints. In *Proc. of ACM Knowledge Discovery and Data Mining (SIGKDD)*, pages 279–288, 2002.

[18] K. Kenthapadi, N. Mishra, and K. Nissim. Simulatable auditing. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, pages 118–127, 2005.

[19] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *Proc. of Very Large Data Bases (VLDB)*, pages 199–210, 2006.

[20] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *Proc. of International Conference on Data Engineering (ICDE)*, 2008.

[21] F. McSherry and K. Talwar. Mechanism design via differential privacy. 2007.

[22] S. U. Nabar, B. Marthi, K. Kenthapadi, N. Mishra, and R. Motwani. Towards robustness in query auditing. In *VLDB*, pages 151–162, 2006.

[23] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data anlysis. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, pages 75–84, 2007.

[24] V. Rastogi, S. Hong, and D. Suciu. The boundary between privacy and utility in data publishing. In *Proc. of Very Large Data Bases (VLDB)*, pages 531–542, 2007.

[25] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(6):1010–1027, 2001.

[26] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[27] X. Zhou, J. Gaugaz, W.-T. Balke, and W. Nejdl. Query relaxation using malleable schemas. In *Proc. of ACM Management of Data (SIGMOD)*, pages 545–556, 2007.

Appendix: Proof of Lemma 1

We will prove the lemma by a reduction from the *maximum 2-satisfiability* (MAX-2-SAT) problem. Specifically, let F be a 2-CNF formula with m clauses on n variables v_i $(1 \le i \le n)$. Given a positive integer k, MAX-2-SAT asks whether there is an assignment of boolean values to v_i, such that at least k clauses in F evaluate to *true*. This problem is NP-complete [16]. In the sequel, if a clause is *true*, we say that it is *satisfied*.

Recall that all sibling tables conform to the same schema. We consider that the schema has n attributes A_1, ..., A_n, all of which have a domain $\{0, 1, 2, 3\}$. Given a 2-CNF formula F, we create a set Q of $6m$ count queries as follows. Let c_j be the j-th ($1 \le j \le m$) clause in F, and assume that c_j involves the α-th and β-th variables v_α and v_β. Define $b_\alpha = 0$ if the negation of v_α appears in c_j, and $b_\alpha = 1$ otherwise. Also define b_β according to v_β in the same manner. We add to Q the following 6 queries:

q_{j1}: SELECT COUNT(*) FROM T
WHERE $A_\alpha = b_\alpha$ AND $A_\beta = b_\beta$

q_{j2}: SELECT COUNT(*) FROM T
WHERE $A_\alpha = b_\alpha$ AND $A_\beta = 1 - b_\beta$

q_{j3}: SELECT COUNT(*) FROM T
WHERE $A_\alpha = 1 - b_\alpha$ AND $A_\beta = b_\beta$

q_{j4}: SELECT COUNT(*) FROM T
WHERE $A_\alpha = 2 + b_\alpha$ AND $A_\beta = 2 + b_\beta$

q_{j5}: SELECT COUNT(*) FROM T
WHERE $A_\alpha = 2 + b_\alpha$ AND $A_\beta = 3 - b_\beta$

q_{j6}: SELECT COUNT(*) FROM T
WHERE $A_\alpha = 3 - b_\alpha$ AND $A_\beta = 2 + b_\beta$

It is important to note that any tuple can satisfy *at most* one of these queries. Repeating the above for all m clauses in F yields totally $6m$ queries. The rest of the proof will establish:

PROPOSITION 1. $S_{L1}(Q) \geq 2k$ *if and only if there is an assignment of boolean values to* v_i $(1 \leq i \leq n)$ *that satisfies at least* k *clauses in* F.

The "only-if" direction. Without loss of generality, assume that c_1, ..., c_k in F are *true* under a certain boolean-value assignment. We build a pair of sibling microdata tables T_1 and T_2 as follows. T_1 has a single tuple t_1, such that for any $1 \leq i \leq n$, $t_1[A_i] = 1$ if v_i = *true*, and $t_1[A_i] = 0$ otherwise. Similarly, T_2 also has a single tuple t_2 such that $t_2[A_i] = 2 + t_1[A_i]$.

Consider clause c_j $(1 \leq j \leq k)$ in F, and the six queries q_{j1}, ..., q_{j6} in Q created from c_j. Let v_α and v_β be the two variables in c_j. Suppose that v_α = *true* and v_β = *false* (the proof for the other cases is similar). Accordingly, $t_1[A_\alpha] = 1$ and $t_1[A_\beta] = 0$. Since c_j evaluates to *true*, either v_α or $\neg v_\beta$ appears in c_j, implying that $b_\alpha = 1$ or $b_\beta = 0$. Hence, t_1 satisfies *one* of q_{j1}, q_{j2}, q_{j3} but violates all of q_{j4}, q_{j5}, q_{j6}. Similarly, it can be verified that t_2 satisfies one of q_{j4}, q_{j5}, q_{j6} but violates all of q_{j1}, q_{j2}, q_{j3}. It means that

$$\sum_{l=1}^{6} \left| q_{jl}(T_1) - q_{jl}(T_2) \right| = 2.$$

Therefore

$$\begin{aligned} S_{L1}(Q) &\geq \sum_{q \in Q} \left| q(T_1) - q(T_2) \right| \\ &\geq \sum_{j=1}^{k} \sum_{l=1}^{6} \left| q_{jl}(T_1) - q_{jl}(T_2) \right| \\ &= 2k. \end{aligned}$$

The "if" direction. As $S_{L1}(Q) \geq 2k$, there exists a pair of siblings T'_3 and T'_4 such that $\sum_{q \in Q} \left| q(T'_3) - q(T'_4) \right| = S_{L1}(Q) \geq 2k$. Recall that T'_3 and T'_4 differ in only one tuple. Denote by t_3 (t_4) the tuple in T'_3 (T'_4) that does not appear in T'_4 (T'_3). Let T_3 (T_4) be a microdata table where t_3 (t_4) is the only tuple. We have

$$\sum_{q \in Q} \left| q(T_3) - q(T_4) \right| = \sum_{q \in Q} \left| q(T'_3) - q(T'_4) \right| \geq 2k. \quad (14)$$

Apparently, for any query q, $q(T_3)$ and $q(T_4)$ equal either zero or one. By Equation 14, it must be that either $\sum_{q \in Q} q(T_3) \geq k$ or $\sum_{q \in Q} q(T_4) \geq k$. Without loss of generality, assume $\sum_{q \in Q} q(T_3) \geq k$. Then,

$$k \leq \sum_{q \in Q} q(T_3) = \sum_{j=1}^{k} \sum_{l=1}^{6} \left| q_{jl}(T_3) \right|. \quad (15)$$

We assign boolean values to v_i $(1 \leq i \leq n)$ as follows:

$$v_i = \begin{cases} \textit{true}, & \text{if } t_3[A_i] = 1 \text{ or } t_3[A_i] = 3 \\ \textit{false}, & \text{otherwise} \end{cases} \quad (16)$$

We will show that the above assignment satisfies at least k clauses of F. For any $j \in [1, m]$, at most one of $q_{j1}(T_3)$, $q_{j2}(T_3)$, ..., $q_{j6}(T_3)$ is one, and the others must be 0. Let J be the set of integers in $[1, m]$, such that for any $j \in J$, $\sum_{l=1}^{6} \left| q_{jl}(T_3) \right| = 1$. By Equation 15, $|J| \geq k$ holds.

For any $j \in J$, consider the j-th clause c_j in F. Again, assume v_α and v_β to be the variables in c_j. As $\sum_{l=1}^{6} \left| q_{jl}(T_3) \right| = 1$, t_3 satisfies one of q_{j1}, ..., q_{j6}. Without loss of generality, assume that t_3 satisfies q_{j5} (the proof for the other cases is similar), namely, $t_3[A_\alpha] = 2 + b_\alpha$ and $t_3[A_\beta] = 3 - b_\beta$. The values of b_α and b_β can independently be either 0 or 1. Regardless of their values, c_j always evaluates to *true*. For example, suppose $b_\alpha = 0$ and $b_\beta = 1$. As $b_\alpha = 0$, we know $t_3[A_\alpha] = 2$, and by Equation 16, v_α has been set to *false*. Furthermore, $b_\alpha = 0$ also suggests that $\neg v_\alpha$ is in c_j, which hence evaluates to *true*.

As in the above reasoning j was chosen to be any integer in J, we have identified at least $|J| \geq k$ clauses which evaluate to *true*.