Query Optimization for Differentially Private Data Management Systems

Shangfu Peng\(^1\), Yin Yang\(^2\), Zhenjie Zhang\(^2\), Marianne Winslett\(^2\), Yong Yu\(^4\)

\(^1\)Department of Computer Science, University of Maryland at College Park, MD, USA
shangfu@cs.umd.edu

\(^2\)Advanced Digital Sciences Center, Singapore
{yin.yang, zhenjie}@adsc.com.sg

\(^3\)Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
winslett@illinois.edu

\(^4\)Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
yyu@cs.sjtu.edu.cn

**Abstract**—Differential privacy (DP) enables publishing the results of statistical queries over sensitive data, with rigorous privacy guarantees, and very conservative assumptions about the adversary’s background knowledge. This paper focuses on the interactive DP framework, which processes incoming queries on the fly, each of which consumes a portion of the user-specified privacy budget. Existing systems process each query independently, which often leads to considerable privacy budget waste and consequently fast exhaustion of the total budget. Motivated by this, we propose Pioneer, a query optimizer for an interactive, DP-compliant DBMS. For each new query, Pioneer creates an execution plan that combines past query results and new results from the underlying data. When a query has multiple semantically equivalent plans, Pioneer automatically selects one with minimal privacy budget consumption. Extensive experiments confirm that Pioneer achieves significant savings of the privacy budget, and can answer many more queries than existing systems for a fixed total budget, with comparable result accuracy.

I. INTRODUCTION

Differential privacy [4] (DP) is one of the strongest schemes for protecting individuals’ privacy in published group statistics. DP ensures it is provably hard for an adversary to infer the presence or absence of any specific record, even if the adversary knows all other DB records. DP has rapidly gained popularity in multiple research fields, including computer networking [19], data mining [8], and medical analysis [30]. A common solution for DP is to inject random noise into statistical query results. The noise scale is controlled by a parameter \(\epsilon\) called the privacy budget. A smaller \(\epsilon\) leads to more noise and stronger privacy in the sense of increased difficulty for the adversary to derive sensitive information.

DP can be offline or interactive. The former answers a pre-defined set of queries and publishes just those answers, while the latter processes incoming queries as they arrive. We focus on the latter, which enables DP-compliant DBMSs with a broad range of applications. One example is to build such a DB for electronic health records, which answers queries about the number of patients suited for a clinical trial. A pharmaceutical company trial needs patients with particular medical conditions and demographics. Before committing to a trial in, say, Singapore, the company must determine whether Singapore has enough patients satisfying the criteria. Since different trials have different patient criteria, the patient count queries cannot be computed in advance. A DP-compliant DBMS fits well here, as it can answer incoming patient count queries while protecting individuals’ privacy.

Figure 1 illustrates the architecture of a DP-compliant DBMS, in which the client issues every statistical query \(q\) to a query executor. The latter then computes the exact result of \(q\) over the DB, and returns a perturbed version of the result to the client. Because different queries may have different accuracy requirements (e.g., a patient count query for a rare disease might need to be answered more accurately than that for a common disease), the client also specifies the required accuracy of \(q\), e.g., in terms of variance \(1\). To satisfy DP, the DBMS maintains a total privacy budget \(\epsilon_{\text{tot}}\) whose initial value is set by the administrator. As we explain in Section II-A, the execution of each query \(q\) consumes a portion of the privacy budget \(\epsilon_q\), which is deducted from \(\epsilon_{\text{tot}}\). If \(\epsilon_q > \epsilon_{\text{tot}}\), \(q\) is rejected; when \(\epsilon_{\text{tot}}\) decreases to zero, no more queries can be answered. Clearly, minimizing \(\epsilon_q\) is critical for the DBMS.

Figure 1: Architecture of a DP-compliant DBMS

Existing DP-compliant DBMSs, e.g., PINQ [16], add noise to the results of each query independently. We observe that often,

---

1 Some existing systems require the client to specify the privacy budget of each query, rather than its accuracy. Our discussions with potential users reveal that it is hard for them to determine the appropriate privacy budget, and specifying the required accuracy is much more intuitive. Similar observations are made in [21].
significant privacy budget can be saved by reusing previous query results to answer a new query. Consider queries \(q_1\) and \(q_2\) in Figure 2a, where the exact answer to \(q_1\) is the sum of those of \(q_1\) and \(q_2\), and the DBMS has already processed \(q_1\) and \(q_2\). Suppose that the noisy answers to \(q_1\) and \(q_2\) are \(R_1\) and \(R_2\), with variance \(v_1=2\) and \(v_2=2\), respectively, according to the client’s requirements. Since the noise injected into each result has mean value 0 [6], \(R_1+R_2\) has variance \(2+2=4\) with respect to the exact answer of \(q_1+q_2\). Hence, if the requested variance for \(q_3\) is \(v_3=4\), we can simply answer \(q_3\) by \(R_1+R_2\) without spending any privacy budget at all.

Now consider a more complicated case with the same queries, and accuracy requirements. \(v_1=2\), \(v_2=2\), \(v_3=2\). \(R_1+R_2\) no longer answers \(q_3\) with sufficient accuracy, and the query executor must retrieve new information from the sensitive data. On the other hand, reusing \(R_1\) and \(R_2\) can still reduce the privacy budget for answering \(q_3\). Figure 2b shows an execution plan \(EP_1\) for \(q_3\) that first adds \(R_1\) and \(R_2\), and then averages \(R_1+R_2\) with the noisy answer of another query \(q'\), which is identical to \(q_3\) except that it has a lower accuracy requirement \(v'_3=4\). The result has variance \((4+4)/2^2=2\), which satisfies \(q_3\). As we explain in Section II-A, the privacy budget consumed by a query is inversely proportional to the square root of its variance. This means that the privacy cost of \(q'\) is only \(\sqrt{1/2}\) of that of \(q_3\). Since \(q'\) is the only operator that requires sensitive information, the total privacy cost of plan \(EP_1\) is significantly smaller than for computing \(q_3\) directly.

Figure 2c shows another plan \(EP_2\) for \(q_3\), which involves the NoiseDown (ND) operator [25]. \(ND\) takes the noisy result of a previous query (e.g., \(q_1\) with \(v_1=2\)), and outputs a more accurate version (e.g., with \(v'_1=1\)). The additional privacy cost of \(ND\) is the difference between the costs of answering the same query with the two different values for noise variance. In our running example with \(v_1=2\), \(v'_1=1\), \(EP_2\) refines \(R_1\) and \(R_2\) to obtain their answers with variance \(v'_1=v'_2=1\), and adds them to answer \(q_3\) with variance \(v'_1+v'_2=2\). As we explain in Section III, the privacy cost of \(EP_2\) in this particular case is higher than for \(EP_1\); the reverse is true in other settings, e.g., if \(v_1=1\), \(v'_1=1\). This means that the relative performance of different plans depends on the workload.

There are two main challenges in realizing an effective optimizer for DP-compliant DBMSs. First, there is a vast search space for possible execution plans for answering an incoming query \(q\), and it is nontrivial to select the best plan. The example in Figure 2, for instance, involves only two past queries and two simple plan types; yet the optimizer must determine several parameters (i.e., \(v'_3\) in \(EP_1\), \(v'_1\) and \(v'_2\) in \(EP_2\)), and select the plan that minimizes privacy budget usage. As we show in the paper, more general and effective plans are also more complex, requiring sophisticated algorithms for plan building and selection. The second challenge lies in finding past query sets that can be used to build a plan for the new query \(q\). There can be a large number of past queries with numerous possible combinations; meanwhile, an obvious combination of past queries that can answer \(q\) may not exist. For example, in Figure 2, if we modify \(q_1\)’s selection condition to “age between 10 and 15”, \(q_3=q_1\), no longer holds, and there is no apparent way to answer \(q_1\) with \(q_1\) and \(q_2\). We face these challenges with a novel solution Pioneer, an effective and generic query optimizer for DP-compliant DBMSs. Pioneer selects the best plan based on rigorous analysis of various plan types, and creates opportunities for re-using past queries by exploiting unique properties of query processing under DP. Further, Pioneer incurs negligible computation costs. For simplicity, we present Pioneer for linear counting queries [14], a fundamental task in DP. Nevertheless, Pioneer applies to a wider class of queries that satisfy its assumptions. Extensive experimental studies demonstrate that Pioneer achieves significant privacy budget savings, and thus can answer significantly more queries than existing methods.

In the following, Section II reviews related work, Sections III and IV present the plan building and past query selection modules, respectively. Section V presents a thorough experimental evaluation. Section VI concludes the paper.

II. RELATED WORK

Section II-A provides necessary background on DP. Section II-B overviews query processing techniques in DP systems.

A. Differential Privacy Preliminaries

Recall from Section I that DP guarantees it is hard for the adversary to infer the presence or absence of any individual record, even if she knows the details of all remaining records. DP achieves this by requiring the (noisy) query results to follow similar probability distributions for every two neighbor databases \(DB_1\) and \(DB_2\). \(DB_1\) and \(DB_2\) are neighbors iff \(DB_2\) can be obtained by adding or removing one record to/from \(DB_1\). Let \(q\) be an aggregate query, \(s\) a possible noisy answer to \(q\), and \(q(DB)\) a random variable representing the answer given to the user. The most commonly used version of DP, called \(\epsilon\)-differential privacy (\(\epsilon\)-DP) [4], requires the following:

\[
Prob(q(DB_1) = s) \leq e^\epsilon \cdot Prob(q(DB_2) = s).
\]  

(1)

Intuitively, \(\epsilon\)-DP ensures that the change in probability caused by adding/removing a record is bounded by a constant factor \(e^\epsilon\). \(\epsilon\)-DP is composable [4], meaning that a system satisfies \((\sum_{i=1}^m \epsilon_i)\)-DP after answering queries \(q_1, ..., q_m\) independently with \(\epsilon_1, ..., \epsilon_m\)-DP guarantees, respectively. This property is the foundation for an interactive DP-compliant database, in which each query consumes a portion of the total privacy budget \(\epsilon_{tot}\). After \(\epsilon_{tot}\) depletes and the database is shut down, composability guarantees that the system satisfies \(\epsilon_{tot}\)-DP.
The Laplace mechanism [6] offers an effective way to achieve \( \epsilon \)-DP, by adding random Laplace noise to the exact query answers. The noise has mean 0, and a scale of \( \Delta \cdot \epsilon \), where \( \Delta \) is the sensitivity of the query \( q \), defined as the maximum possible difference in the actual results of \( q \) on any two neighbor databases. Formally, we have:

\[
\Delta_q = \max_{DB_1, DB_2} |q^{ACT}(DB_1) - q^{ACT}(DB_2)|,
\]

(2)

where \( q^{ACT}(DB) \) denotes the exact answer to \( q \) over \( DB \), and \( DB_1, DB_2 \) are two arbitrary neighbor databases. For instance, the sensitivity of a selection-count query is 1, since adding / removing a record can affect the query result by at most 1. Accordingly, the injected Laplace noise has a scale of \( 1/\epsilon \).

Because the Laplace mechanism injects zero-mean noise into the query results, the variance of the noisy result is the main measure of its error. Let \( v_i \) be the variance of the noisy results for \( q_i \), \( \epsilon_i \) its privacy budget consumption, and \( \Delta_i \) its sensitivity. Properties of the Laplace distribution ensure:

\[
v_i = 2 \cdot (\Delta_i/\epsilon_i)^2, \quad \epsilon_i = \sqrt{2/v_i} \cdot \Delta_i.
\]

(3)

Equation (3) shows that the budget consumption for a query is inversely proportional to the square root of its required variance. In Figure 2, plan \( EP_i \) uses this to reduce the budget cost of \( q_i \), by processing a lower-accuracy version \( q'_i \) of \( q_i \).

The exponential mechanism [20] is another popular method to enforce \( \epsilon \)-DP, e.g., on queries with categorical results. Because certain queries have high sensitivity, previous work has proposed relaxations of \( \epsilon \)-DP, e.g., \( (\epsilon, \delta) \)-DP [5], which guarantees \( \epsilon \)-DP with a certain probability. Mechanisms that comply with \( \epsilon \)-DP also satisfy \( (\epsilon, \delta) \)-DP, but the reverse is usually not true, e.g., methods proposed in [14][17] satisfy \( (\epsilon, \delta) \)-DP, but not \( \epsilon \)-DP. This paper focuses on the stronger \( \epsilon \)-DP definition, though some ideas also apply to \( (\epsilon, \delta) \)-DP.

**B. Differentially Private Query Processing**

Although the Laplace mechanism can answer all types of queries under DP, straightforward applications of this method to practical query workloads often lead to poor result accuracy with given values for \( \epsilon \), or, in our setting with user-specified error levels, considerable waste of privacy budget. Therefore, researchers have proposed numerous algorithms suitable for specific applications. This section focuses on query processing techniques proposed by the database community [28].

**Offline methods.** Xiao et al. [26] investigate range-count queries, and propose an offline solution Privlet, which takes \( \epsilon \) as input and publishes the results of all unit-range counts. The answer to a query \( q \) with a wider range is derived by adding the unit-size counts within \( q \)'s range. Privlet guarantees that the noise variance of any one-dimensional range count query is bounded by \( O(\log^3 n/\epsilon^2) \), where \( n \) is the domain size on which the range selection is performed. Hay et al. [10] tackle the problem that the noisy answers to different queries are often inconsistent [1], and design effective methods to output consistent (and more accurate) results. Refs. [3][22] propose synopsis for multi-dimensional range counts. Chen et al. [2] solve the problem of publishing set-valued data under DP. Ding et al. [7] study optimal data cube publication under DP. Xu et al. [27] and Li et al. [15] use compression techniques to reduce the noise in a differentially private synopsis.

The main limitation of non-interactive methods is that their privacy budget usage is determined by the highest possible required accuracy for a query. Consequently, they are not suitable for applications with very different accuracy requirement for different queries. In practice, an application may only need to answer a small number of queries with high accuracy (e.g., those critical to a clinical trial), and the remaining ones with low accuracy. In an extreme case, even if only one query needs to be answered with high accuracy \( v_{\text{high}} \), a non-interactive method still needs to build a synopsis whose accuracy lower bound exceeds \( v_{\text{high}} \), leading to tremendous waste in privacy budget. Interactive methods, reviewed next, handle such situations much more effectively.

**Interactive methods.** Existing interactive DP systems, e.g., PINQ [16], answer each incoming query independently, without reusing previous query results. Li et al. [13][14] investigate the processing of a given batch of linear counting queries, and propose methods that process them more efficiently than answering them individually. Queries in different batches are still processed independently. Yuan et al. [29] targets the same problem, and propose improved solutions. Rastogi and Nath [23] study the processing DP queries over time series DBs, where queries have very high sensitivity. Machanavajjhala et al. [18] apply DP to social recommendation systems. These methods can be used in combination with the proposed approach. Interactive DP work in the theory community (e.g., [9][24]) does not discuss the opportunity of reusing existing query results, either.

Xiao et al. [25] develop techniques that minimize the total relative error of a given batch of queries. The contributions of [25] include the ND operator mentioned in Section I. Specifically, suppose that query \( q_i \) has been answered with a noise variance of \( v_i \); the ND operator refines the accuracy of \( q_i \) to a new variance \( v_i' < v_i \) with a privacy budget as follows.

\[
\epsilon_i^{ND} = \left( \sqrt{2/v_i'} - \sqrt{2/v_i} \right) \cdot \Delta_i.
\]

(4)

The ND operator does not waste any budget, because the privacy cost of first answering \( q_i \) with variance \( v_i \), and then refining it to variance \( v_i' \) with ND is the same as answering \( q_i \) directly with variance \( v_i' \). ND achieves this by adding noise with a complicated distribution that depends on both the actual result of \( q_i \) and the previous noisy result with variance \( v_i \). The method of [25] relies on unique properties of the Laplace distribution; consequently, ND cannot refine a composite result, e.g., the sum of multiple noisy results, which no longer follow the Laplace distribution. For this reason, In Figure 2, one cannot put ND after the sum operator. Finally, Kifer and Machanavajjhala [11][12] discuss the limitations of DP in social network publication, suggesting that an even stronger privacy definition might be needed for certain applications.

**III. BUILDING AN EXECUTION PLAN**

This section describes how Pioneer constructs the best query plan for an incoming query \( q \) given a set of relevant queries \( Q \).
Pioneer requires (i) all past queries operate on the same dataset, and contain the same aggregate clause, e.g., COUNT, SUM, etc. as the new query; (ii) noise added to each query in $Q$ follow zero-mean, Laplace distributions. Besides these, Pioneer does not assume any specific query type. For simplicity, we use linear counting queries in our presentation, and explain the difference in processing other query types whenever necessary. As in most existing work on DP, Pioneer measures the accuracy of a query result by its noise variance.

Section IV explains how to find $Q$ (i.e., answerwability of a new query with past queries) and divide it into subsets $Q = \cup_{i \in [m]} Q_i$, such that for each $Q_i$, the exact answer to $q$ is a linear combination of the exact answers to the queries in $Q_i$, for every DB instance. In our running example in Figure 2, we have $q = q_1$, and $Q = \{q_1, q_2\}$, i.e., the linear combination $q_1 + q_2$ is equivalent to $q$. More formally, if $Q_i$ contains queries $q_{i1}, \ldots, q_{i(m_i)}$, then there must be a constant $c_{i,j}$ associated with each query $q_{i,j}$, such that:

$$q = \sum_{j=1}^{m_i} (c_{i,j} q_{i,j}), \forall 1 \leq i \leq n.$$  

Note that $c_{i,j}$ can be negative, e.g., $q = q_{1,2} + q_{2,1} - q_{1,3}$. As explained in Section IV, negative coefficients greatly increase the flexibility for composing $q$ with past queries. To avoid notational overload, we set all coefficients $c_{i,j}$ to 1 until the end of Section III-C:

$$q = \sum_{j=1}^{m_i} q_{i,j}, \forall 1 \leq i \leq n.$$  

Until the end of Section III-C, we consider only queries with sensitivity 1, e.g., count queries, as explained in Section II-A. Until Section III-D, we require that each query occur in at most one subset $Q_i$ of $Q$.

In the following, Sections III-A to III-D presents 4 types of query plans: *top branching (TB) plans*, *leaf refining (LR) plans*, TB-LR hybrid plans, and composite plans. Among these, TB and LR plans show the basic idea of reusing results from past queries. For instance, plans $EP_1$ and $EP_2$ in Figure 2 are example TB and LR plans, respectively. The hybrid plan generalizes these two basic plan types, and performs at least as well as these two basic plan types. Table I summarizes symbols used throughout the paper.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q, v$</td>
<td>Incoming query and its specified noise variance</td>
</tr>
<tr>
<td>$Q$</td>
<td>Set of relevant previous queries</td>
</tr>
<tr>
<td>$Q_i, m_i$</td>
<td>$i$-th partition of $Q$ and the number of queries therein</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of partitions of $Q$</td>
</tr>
<tr>
<td>$q_{i,j}, v_{i,j}$</td>
<td>A query in $Q_i$ and its noise variance</td>
</tr>
<tr>
<td>$w_i, w_i'$</td>
<td>Weight associated with partition $Q_i$ in the TB and LR plans</td>
</tr>
<tr>
<td>$q', v'$</td>
<td>New query processed in a TB plan, and its noise variance</td>
</tr>
<tr>
<td>$v_{i,j}'$</td>
<td>Refined noise variance of query $q_{i,j}$ in an LR plan</td>
</tr>
<tr>
<td>$\epsilon_{plan}$</td>
<td>Privacy budget consumption of a plan, e.g., TB, LR, hybrid</td>
</tr>
</tbody>
</table>

### A. TB Plans

Figure 3 illustrates the structure of a TB plan. First, for each partition $Q_i$ of $Q$, the plan computes the sum of $Q_i$’s results, using Equation (6). Additionally, the TB plan answers $q = q_i$, but with a lower accuracy $v'$ than the required accuracy $v$ for $q$. These answers are fed into a single weighted average operator AVG, whose output answers $q$ with sufficient accuracy.

The privacy budget consumption of the TB plan equals to that of answering the query $q'$, which, in turn, is determined by the required accuracy $v'$ of $q'$. Hence, in the following we derive the appropriate plan parameters to minimize $v'$. These parameters are the weights assigned to the $n+1$ inputs of the AVG operator. Suppose that all these weights sum up to 1. Let $w_i (1 \leq i \leq n)$ denote the weight assigned to the output of the $i$-th sum operator. Then, the weight assigned to $q'$ is $1 - \sum_{i=1}^{n} w_i$.

Let $v_{i,j}$ be the noise variance of query $q_{i,j}$, and $v_{i,*}$ be the variance for the output of the $i$-th sum that covers query set $Q_i$. Since the noise for each query follows an independent, zero-mean Laplace distribution, $v_{i,*}$ is simply the sum of variance for queries in $Q_i$, i.e., $\sum_{j=1}^{m_i} v_{i,j}$. Therefore, we have:

$$v = \sum_{i=1}^{n} w_i v_{i,*} + \left(1 - \sum_{i=1}^{n} w_i\right)^2 v'.$$  

Figure 3: Structure of a TB plan

The minimum value of $v'$ is obtained, when the partial derivative of Equation (7) with respect to each $w_i$ equals 0. Accordingly, we solve for the optimal values for $w_i$ and $v'$, and obtain the following:

$$w_i = \frac{v}{v_{i,*}} (1 \leq i \leq n), \quad v' = \frac{1}{v - \sum_{i=1}^{n} \frac{1}{v_{i,*}}}.$$  

If the denominator in Equation (8) yields a non-positive value, then $q'$ is unnecessary, and the rest of the plan answers $q$ with sufficient accuracy. Finally, combining Equation (8) and our simplifying assumption that each query has sensitivity 1 (which is removed in Section III-C), we obtain privacy budget cost of the TB plan, as follows.

$$\epsilon_{TB} = \sqrt{\max \left(0, \frac{2}{v} - \sum_{i=1}^{n} \frac{2}{v_{i,*}}\right)}.$$  

In the running example in Figure 2, according to the above analysis, the privacy cost for plan $EP_1$ for the setting $v_1 = v_2 = v_3 = 2$ is $\sqrt{1/3}$, which is reached when $v_3 = 4$. Similarly, for the setting $v_1 = 1, v_2 = v_3 = 2$, the minimum budget consumption is $\sqrt{1/3}$, with $v_3 = 6$. In both cases, the privacy cost is significantly lower than that of answering $q$ directly, i.e., 1 according to Equation (3).
B. LR Plans

Figure 4 shows the LR plan, which uses the NoiseDown (ND) operator described in Section II-B, as well as the sum and AVG operators seen earlier. The accuracy of each previous query result \( q_{ij} \) is refined with an ND, and then fed into its corresponding sum. Note that as explained in Section II-B, one cannot place an ND downstream of the sum or AVG operators.

The LR plan consumes privacy budget through the ND operators. Hence, in the following we focus on minimizing the sum of privacy cost of all NDs in the plan. According to Equation (4), the budget cost of an ND is determined by the noise variance of its input and output. Let \( \nu'_{ij} \) (referred to as the refined variance) be the variance of the output of the ND for query \( q_{ij} \). Similar to the TB plan, we define \( \nu'_{i} = \sum_{j=1}^{m_i} \nu'_{ij} \), and use \( w'_{i} \) be the normalized weight assigned to the \( i \)-th input to AVG. Similar to the analysis in Section III-A, the best weights are as follows.

\[
w'_{i} = v / \nu'_{i}, \quad (1 \leq i \leq n), \quad v = 1 \sum_{i=1}^{n} \frac{1}{\nu'_{i}}.
\] (10)

By Equation (4), for queries with sensitivity 1, the total privacy cost of the LR plan is:

\[
e_{LR} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \left( \frac{2}{\nu'_{ij}} - \frac{2}{\nu_{ij}} \right). 
\] (11)

The first derivative of \( e_{LR} \) with respect to variance \( \nu'_{ij} \) is then:

\[
\frac{\partial e_{LR}}{\partial \nu'_{ij}} = \frac{-1}{\nu'_{ij} \cdot (2 \nu'_{ij})^{1/2}}, \quad 1 \leq i \leq n, 1 \leq j \leq m_i.
\] (12)

Equation (12) suggests that the marginal privacy cost with respect to \( \nu'_{ij} \) grows as \( \nu'_{ij} \) decreases. Intuitively, this means we give up more privacy to refine a query result that is already quite accurate, than to refine another that is less accurate. Meanwhile, reducing the variance of any of queries in the same partition \( Q_i \) by a constant \( \delta \) has the same effect on the plan, i.e., decreasing \( \nu_{ij} \) by \( \delta \). Without loss of generality, we assume that queries in \( Q_i \) are sorted in ascending order of their original accuracy, i.e., \( \nu_{ij} \geq \nu_{ij+1} \), for all \( i \) and \( j \). It follows that the best LR plan should first refine the most inaccurate query in \( Q_i \), i.e., \( q_{ij}, \) to the level of the second most inaccurate query, i.e., \( v_{ij} \), and then reduce both of them to \( v_{ij} \), and so on. These observations are summarized in the following lemma.

**Lemma 1:** Suppose that we are given the value for \( \nu'_{ij} \), in each partition \( Q_i \) of the best LR plan, and \( \nu_{ij} \geq \nu_{ij+1} \), for all \( 1 \leq j \leq m_i \). If \( \nu'_{ij} \leq \nu_{ij} \), then the best LR plan satisfies \( \nu'_{ij} = \nu_{ij} \), for \( 1 \leq j \leq m_i \). Otherwise, there exists \( 1 \leq l < m_i \) such that:

\[
\sum_{j=1}^{l-1} \nu'_{ij} + \sum_{j=l}^{m_l} \nu'_{ij} \leq \nu'_{ij} < \sum_{j=1}^{l-1} \nu_{ij+1} + \sum_{j=l+1}^{m_l} \nu_{ij+1}.
\]

Further, the best LR plan satisfies \( \nu'_{ij} = \nu_{ij} \), for \( 1 \leq j \leq l \), and \( \nu'_{ij} = \nu_{ij} \), for \( l + 1 \leq m_i \).

Next we focus on the computation of the best values for \( \nu'_{ij} \). Consider \( \nu \) as a function of \( \nu'_{ij} \). By Equation (10), the first derivative of \( \nu \) with respect to each \( \nu'_{ij} \), reflects the marginal accuracy benefit of reducing \( \nu'_{ij} \), which is:

\[
\frac{\partial \nu}{\partial \nu'_{ij}} = \frac{\partial \nu}{\partial \nu_{ij}} = v^2 \cdot \frac{2 \nu'_{ij} \cdot (2 \nu'_{ij})^{1/2}}{(\nu'_{ij})^2}.
\] (13)

This equation implies that the marginal accuracy benefit of reducing \( \nu'_{ij} \) increases as \( \nu'_{ij} \) decreases. But as discussed above, the marginal privacy cost for reducing \( \nu'_{ij} \) also increases as \( \nu'_{ij} \) decreases. This conflict makes finding the best LR plan rather complicated. We employ a heuristic approach, defining the score of a query \( q_{ij} \) as the ratio of its marginal accuracy benefit and marginal privacy cost:

\[
\text{score}_{ci} = -\frac{\partial \nu / \partial \nu'_{ij}}{\partial e_{LR} / \partial \nu'_{ij}} = \frac{v^2 \cdot \nu_{ij} \cdot (2 \nu_{ij})^{1/2}}{(\nu'_{ij})^2}.
\] (14)

Figure 5 gives our CLR algorithm for building an LR plan. CLR initializes the refined variance \( \nu'_{ij} \) of each query \( q_{ij} \) to the original variance \( \nu_{ij} \) (line 2), and calculates its score accordingly. Then CLR iterates (lines 4-16), preserving the condition in Lemma 1 at all times. Each iteration starts by selecting a partition \( Q_i \) that contains queries with the highest score (line 5). The marginal accuracy benefit defined in Equation (13) is the same for all queries in \( Q_i \); the query \( q_{ij} \) in \( Q_i \) has the lowest marginal privacy cost, by Lemma 2. After that, CLR attempts to refine the accuracy of \( q_{ij} \) and all other queries \( q_{ij} \) with the same refined accuracy as \( q_{ij} \). The target is to reduce their variance to \( \nu''_{ij} \), i.e., the query with accuracy just above \( q_{ij} \) in \( Q_i \) (line 11). If such refinement leads to a higher output accuracy than required for \( q \), CLR instead computes \( \nu''_{ij} \) using \( v \) and Lemma 2 (lines 6-9 & 12-14). Finally, CLR updates \( \nu'_{ij} \), and the scores of refined queries, then goes to the next iteration. CLR ends when the LR plan’s output satisfies accuracy requirement \( v \).

Regarding computational overhead, each CLR iteration takes constant time. The number of iterations is bounded by the number of queries in \( Q \), since in every iteration but the last, the algorithm selects a group of queries with the same refined variance, and decreases their variance to the original variance of a target query (line 11); further, the target query is different in each iteration. Therefore, the running time of CLR is linear in the number of past queries in \( Q \). In Figure 2, \( E_P \) is an LR plan. For the setting \( v_1 = v_2 = v_3 = 2 \), CLR returns \( v'' = v_3 = 1 \), using privacy budget 2 (\( 0.57 \)). However, this is higher than that of \( E_P \) (\( 0.57 \)). On the other hand, for the setting \( v_1 = 1, v_2 = 2 \), the budget consumption of \( E_P \) is \( 0.41 \), reached when \( v'' \) is lower than the privacy budget cost of \( E_P \) (\( 0.57 \)). These facts confirm the need for an effective query optimizer.
CLRP($Q, v_i, q, v$) // Create the LR plan
// Input: $Q, v_i$: set of previous queries and their result noise variance
// $q, v$: incoming query and its required result variance
// Output: $v_i'$: the refined noise variance for each query in $Q$
1. For each pair of $i, j$, $1 \leq i \leq m_i$, $1 \leq j \leq m_i$
2. Initialize $v_i'=v_{ij}$
3. Calculate $score_{ij}$ according to Equation (14)
4. Repeat
5. Select a partition $Q$ that maximizes $score_{ij}$
6. If all queries in $Q_i$ have the same refined variance
7. Solve $v_i'$ from Equation (10)
8. Set the refined variance of all queries in $Q_i$ to $v_i'$
9. Goto line 11 // break the repeat-until loop
10. Let $q_{ij}$ be the query with maximum $j$ satisfying $q_i'=v_{ij}'$
11. Refine the variance of queries $q_{ij}$, $q_{ij}^k$ to $v_{ij}'$
12. If the output of the LR plan has variance smaller than $v_i$
13. Roll back the refinement step in Line 11
14. Solve $v_i'$ from Equation (10), and set the refined variance of $q_{ij}$, according to Lemma 2
15. Update the score for each query in $Q_i$
16. Until Equation (10) is satisfied
17. Return $v_i'$ for all $1 \leq i \leq m_i$, $1 \leq j \leq m_i$

Figure 5: CLRP algorithm: Create an LR plan

C. Hybrid Plans

As shown in Figure 6, the hybrid plan integrates the TB plan idea of issuing a lower-accuracy version of $q$, plus the LR plan idea of refining past queries with the ND operator. Specifically, let $v_i$ and $v_i'$ be the original and refined noise variance of query $q$, respectively, and $v_i', v_i'$ be the noise variance of $q$, $q'$, and the output of the $i$-th sum operator, respectively. Combining the analysis of the TB and LR plans, we have:

$$v = \sum_{i=1}^{n} \left( (w_i')^2 \cdot v_i' \right) + \left( 1 - \sum_{i=1}^{n} w_i' \right)^2 \cdot v'. \quad (15)$$

where the best value for the weight $w_i'$ is $w_i'=\Delta_{q_i}/v_i'$ ($1 \leq i \leq n$).

Let $\epsilon_{hybrid}$ denote the total privacy cost of the hybrid plan:

$$\epsilon_{hybrid} = \sqrt{2/v'} + \sum_{i=1}^{n} \sum_{j=1}^{m_i} \left( \frac{2/v_i'}{2/v_i'} - \frac{2/v_{ij}}{2/v_{ij}} \right). \quad (16)$$

Since the hybrid plan essentially outputs the weighted average of new query $q'$ and an LR sub-plan, Lemma 2 also holds. The hybrid plan creation algorithm CHP, shown in Figure 7 resembles the CLRP algorithm (Figure 5) for creating the LR plan. Similarly, similar to the case of LR plans, we define the score for each $v_i'$, $1 \leq i \leq m_i$, $1 \leq j \leq m_i$, as follows.

$$score_{ij} = \frac{v_{ij}' \cdot (v_i')^{1/2} - \partial \phi / \partial v_{ij}}{(v_i')^{1/2} \cdot \partial \phi / \partial v_{ij}'} \cdot \epsilon_{hybrid} / \partial v_{ij}.$$ \quad (17)

CHP initializes and updates the values for the $v_i'$'s in the same way as CLRP (lines 1 and 7). The main difference between the two algorithms is that in each iteration, CHP calculates the noise variance $v'$ for new query $q'$, as well as the total privacy cost $\epsilon_{hybrid}$ for the entire plan (lines 4-6, 9-10). The final result of CHP is the parameters that minimize the budget over all iterations (line 11). Similar to algorithm CLRP, the computational cost of CHP is linear to the cardinality of $Q$.

CHP($Q, v_i, q, v$) // Create a Hybrid Plan
// Input: $Q, v_i$: set of previous queries and their result noisy variance
// $q, v$: incoming query and its required result variance
// Output: $v_i'$: refined noise variance for each query in $Q$
1. Perform lines 1-3 of algorithm CLRP
2. Initialize $\epsilon_{min} = +\infty$
3. Repeat
4. Solve $v'$ from Equation (15)
5. Compute $\epsilon = \epsilon_{hybrid}$ according to Equation (16)
6. If $\epsilon < \epsilon_{min}$, set $\epsilon_{min} = \epsilon$, and save the plan parameters
7. Same as lines 5-15 of algorithm CLRP
8. Until Equation (10) is satisfied
9. Set $v' = +\infty$, and compute $\epsilon = \epsilon_{hybrid}$ according to Equation (11)
10. If $\epsilon < \epsilon_{min}$, set $\epsilon_{min} = \epsilon$, and save the plan parameters
11. Return the set of $v_i'$ ($1 \leq i \leq m_i$, $1 \leq j \leq m_i$) and $v'$ that achieve $\epsilon_{min}$

Figure 7: CHP algorithm: Create a Hybrid Plan

Note that the first iteration of CHP computes the best parameters of a TB plan, whereas the last one (lines 9-10) obtains the best LR plan. Hence, the hybrid plan subsumes the previous two plan types. Table II demonstrates an example of CHP, with parameters $v=1$, $m=2$, $m_i=m_j=2$, $v_{ij}=3$, $v_{ij}=2$, $v_{ij}=1$. In each iteration, the table shows the refined variance $v'$ for each past query $q'$, and the plans’ total privacy budget cost. In iteration 1, $v_{ij}'=v_{ij}$ for all pairs $i, j$. In iteration 2, CHP reduces the value of $v_{ij}'$, which has the higher score according to Equation (17), whose new value is underlined in the table. In the next two iterations, CHP refines first $v_{ij}'$, then $v_{ij}'$ and $v_{ij}'$. Finally, CHP reports minimum $\epsilon_{hybrid}=0.97$, obtained in iteration 1, along with the corresponding plan parameters.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$v_{ij}'$</th>
<th>$v_{ij}'$</th>
<th>$v_{ij}'$</th>
<th>$v_{ij}'$</th>
<th>$v$</th>
<th>$\epsilon_{hybrid}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>15/7</td>
<td>0.97</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>10/3</td>
<td>1.18</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1.30</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+\infty</td>
<td>1.43</td>
</tr>
</tbody>
</table>

Table II: Example of hybrid plan creation

Pioneer can build a hybrid plan for queries of arbitrary sensitivity, where $q$ is an arbitrary linear combination of existing queries (Equation (5)). Let $\Delta$ (resp. $\Delta_j$) be the sensitivity of query $q$ (resp. $q_j$). Since $q$ differs from $q'$ only in its required accuracy, the sensitivity of $q'$ is also $\Delta$. The privacy cost of a hybrid plan is:

$$\epsilon_{hybrid} = \sqrt{2/v'} + \Delta + \sum_{i=1}^{n} \sum_{j=1}^{m_i} \left( \frac{2/v_{ij}}{2/v_{ij}} - \frac{2/v_{ij}}{2/v_{ij}} \right). \quad (18)$$

Meanwhile, it follows from Equation (5) that the total refined variance of queries in partition $Q$, is $v_i'=\sum_{j=1}^{m_i} (c_{ij}^2 \cdot v_{ij}')$. The remaining analysis and parameter computation for this general case proceeds by simply substituting this definition for $v_i'$, and Equations (18) for (16), in the hybrid plan discussion.
D. Composite Plans

Now we drop the assumption that any two different query partitions are disjoint, and handle that situation using a composite plan, containing multiple sub-plans. Figure 8a shows an example with four previously-answered range-counts \( q_1 \ldots q_4 \) and a new one \( q \). The horizontal line associated with each query shows its range selection. For instance, the range of \( q_1 \) is \([25-50]\). Clearly, \( q = q_1 \cup q_2 \cup q_3 = q_1 \cup q_4 \), so let \( Q_1 = \{ q_1, q_2, q_3 \} \) and \( Q_2 = \{ q_1, q_4 \} \). With the methods from the previous subsections, only one of \( Q_1, Q_2 \) can be used to answer \( q \), since both contain \( q_1 \). Thus, the random noise added to the results of \( q_1 + q_2 + q_3 \) and \( q_1 + q_4 \) are not independent. Consequently, if we feed both \( Q_1 \) and \( Q_2 \) to the AVG operator used in all three execution plans, the noise variance of its output becomes hard to compute, and our previous analysis, e.g., Equations (7), (15), no longer hold.

![Composite Plan Example](image)

Figure 8: Example of a composite plan

On the other hand, in the example, the difference between \( q \) and \( q_1 \) can be expressed in two independent linear combinations, i.e., \( q - q_1 = q_2 - q_3 = q_4 \). This means that \( q - q_1 \) can be answered with a hybrid plan with query partitions \( \{ q_2, q_3 \} \), \( \{ q_4 \} \). Figure 8b displays a plan \( EP \) based on this idea, which contains an LR sub-span \( SP \) that answers \( q - q_1 \) using query sets \( \{ q_2, q_3 \} \), \( \{ q_4 \} \). Meanwhile, the backbone of \( EP \) is a hybrid plan, with \( SP \) (involving queries \( q_2 \), \( q_3 \)) as one input and \( q_1 \) as the other, and independent, input. The hybrid plan here differs slightly from the one shown in Figure 6, in that the former does not have an ND operator on the SP input. This is because the ND operator can only take a simple query as input, not the results of a sub-plan, as explained in Section II-B.

Figure 9 illustrates the general structure of a composite plan, which resembles the TB plan in Figure 3, except that each input in the composite plan is the output of a sub-plan \( SP_i \). In particular, each sub-plan can be a single query (possibly refined by an ND operator); a simple TB, LR, or hybrid plan; or (recursively) a composite plan. For instance, in Figure 8b, we have \( n = 1 \), \( SP_1 = q_1 \), and \( SP_2 = SP \).

![Composite Plan Structure](image)

Figure 9: Structure of a composite plan

It remains to clarify the computation of (i) the best parameters for a given composite plan structure, and (ii) the construction of the best plan from the set of past queries. We first focus on the former. Let \( v, v' \), and \( v'_{SP} \) be the noise variance of incoming query \( q \), query \( q' \), and the output of sub-plan \( SP_i \), respectively. Similar to our previous analysis, the best weights in the topmost AVG operator lead to the following equation.

\[
v = \frac{1}{\text{comp}} \sum_{j=1}^{m} v'_{SP,j}
\]

Each \( v'_{SP,j} \) is also recursively calculated with its inputs, using a similar analysis. For instance, in the composite plan shown in Figure 8b, let \( v'_1, v'_4 \) be the refined variance of queries \( q_1 \), \( q_4 \), and \( v_{SP} \) be the output variance of \( SP \). If \( v = 1/\left(1/v'_1 + 1/v'_4\right) \), then \( v_{SP} = 1/(1/v'_1 + 1/v'_4) \). The privacy budget cost of the composite plan is the sum of the costs for answering new query \( q' \) and for each of the sub-plans. The following theorem states that in the best composite plan, there is no new query in a sub-plan.

**Theorem 1:** The best composite plan issues at most one new query \( q' \), which feeds into the top-level AVG operator.

**Proof (sketch):** For any composite plan \( EP \) in which a sub-plan \( SP \) issues a new query \( q_4 \), we prove that a better plan \( EP' \) exists with lower privacy budget cost. According to the composite plan structure, the parent operator of \( SP \) is a sum, denoted by \( opSUM \), and the parent of \( opSUM \) is an AVG operator, denoted by \( opAVG \). \( EP' \) is identical to \( EP \), except that \( EP' \) issues a new query \( q_4 \) instead of \( q_4 \), and \( q_4 \) feeds into \( opAVG \). Let \( v_{SP}, v'_{SP} \) be the noise variance of \( SP \) in \( EP \) and \( EP' \), respectively, and \( v_i \) be the noise variance of \( q_i \). Similar to the analysis of the TB plan, we obtain:

\[
v_{SP} = \frac{1}{1/v'_{SP}} + \left(1/v_3\right)
\]

Let \( v_{SUM} \), \( v'_{SUM} \) be the noise variance of \( opSUM \) in \( EP \) and \( EP' \), respectively, and \( v_i \) be the noise variance of \( q_i \). Because the two plans differ only in \( SP \), \( opAVG \) has the same output variance in \( EP \) and \( EP' \). Therefore, we have:

\[
v_{SUM} = \frac{1}{1/v'_{SUM}} + \left(1/v_1\right)
\]

Combining Equations (20), (21), and the fact that \( v_{SUM} = v'_{SUM} - v'_{SP} \), we obtain:

\[
v'_{SUM} = \frac{1}{1/v'_{SUM}} + \left(1/v_1\right) = v'_{SP} - \frac{1}{1/v'_{SP}} + \left(1/v_3\right)
\]

Because \( v'_{SUM} = v'_{SP} \), it follows from Equation (22) that \( v_3 < v_1 \). Thus, the privacy cost of \( EP' \) is lower than that of \( EP \).

Let \( \epsilon_{comp} \) be the total privacy budget of the composite plan, \( v_1 \) (resp. \( v' \)) the original (resp. refined) variance of the \( i \)-th query, and \( v' \) the variance of \( q' \). According to Lemma 3, we have:

\[
\epsilon_{comp} = \sum_{i=1}^{n} \left(\sqrt{2/v'_i} - \sqrt{2/v_i}\right) + \sqrt{2/v'}
\]

Based on Equations (19) and (23), we derive the marginal accuracy benefit and privacy cost for decreasing \( v' \):
In the example of Figure 10, suppose that $Q_1, Q_2$ are already sorted by total variance. We initialize $EP$ with $Q_1$. Then, we merge $EP$ with $Q_2$ by creating a new sub-plan that averages $q_2+q_3$ (which are in $EP$ but not in $Q_2$), and $q_4$ ($Q_3/EP$), which leads to the plan in Figure 8b. Finally, $Q_3$ and $EP$ cannot be merged, as $EP/EP3 = \{q_1, q_2, q_4\}$ are scattered across two sub-plans. Consequently, $Q_3$ is discarded, and CCP returns the current $EP$.

IV. SELECTION OF PAST QUERIES

This section deals with the selection of past query set $Q$ to use to build an execution plan for a new query $q$. In the following, we assume all past queries satisfy the requirements mentioned in Section III: that they operate on the same dataset as $q$, contain the same aggregate as $q$, and zero-mean, Laplace noise is injected into their results as noise. Queries that do not satisfy these conditions are simply ignored. Recall that $Q$ is divided into subsets, $Q_1, \ldots, Q_m$ each of which contains queries with a linear combination equivalent to $q$. A brute force approach to computing $Q$ is to enumerate every possible combination of past queries, and verify whether they can be linearly transformed to answer $q$. This method incurs high computational cost. More importantly, direct combinations of existing queries that can answer $q$ may be rare, and may not exist at all, especially when the query selections are performed on a domain with many values or high dimensionality.

Instead, Pioneer employs a novel approach that leverages special properties of $\epsilon$-DP query processing. This technique takes linear time to the number of past queries; further, it ensures that all past queries can potentially be used. Before describing how to pick $Q$, we first explain how a past query $q_i$ can contribute to the processing of $q$. Assume that the required noise variance of $q_i$ is $v_i$ and $v_i$, respectively. Figure 12 illustrates all possible relationships between the selection conditions of $q_i$ and $q_i$, i.e., disjoint, overlapping, or one containing the other. (Although we use one dimensional range selections for ease of presentation, the concepts and methods described here apply to any selection conditions). When $q_i$ and $q$ are disjoint (Figure 12a), it may seem that $q_i$ is unrelated to $q$. However, according to properties of $\epsilon$-DP [6][16], answering $q_i$ alone consumes the same amount of privacy budget as processing both $q_i$ and another query $q$, with accuracy requirement $v_i$. Accordingly, Pioneer generates such a new query $q_s$ without consuming any privacy budget, and adds $q_s$ to $Q$ as a singleton.

Likewise, when $q$ contains $q_i$ (Figure 12b), Pioneer creates a new query $q_s$ (without privacy cost) with selection $q'$ and required variance $v_i$ and adds subset $ \{q_i, q_s\}$ to $Q$. During plan
computation, refining $q_i$ with an ND operator automatically refines $q_i$ to the same accuracy (and vice versa), without additional privacy budget usage. Therefore, the marginal accuracy benefit of refining $q_i$ (or $q_s$) is doubled during plan creation. When $q_s$ contains $q$ (Figure 12c), it is no longer possible to issue a "free" query that answers $q$ either directly or in combination with $q$. On the other hand, recall that all execution plans for $q_s$ except for the LR plan, issue a query $q' \sim q$ with required variance $v' \sim v$. For such plans, Pioneer additionally processes a query $q_i$ with selection $q_i$ and required variance $v'$. Answering $q'$ and $q_i$ consumes the same privacy budget as processing $q'$ alone. Meanwhile, since the linear combination $q_i - q$ answers $q$, Pioneer adds $\{q_i, q_s\}$ to $Q$. Note that the coefficient associated with a query in $Q$ can be negative in Equation (5). Unlike other queries, $q_i$ cannot be refined with an ND operator. Further, the required variance of $q_s$, i.e., $v'$, is unknown beforehand. To handle this, the only (minor) change in TB plan computation is that Equation (8) now involves $v'$ on both sides of the equation, and Pioneer solves for $v'$ there instead of obtaining $v'$ directly from its right hand side. Similar changes apply to the computation of the hybrid and composite plans as well. Figure 12d shows the situation where $q_i$ overlaps $q_s$ but neither contains the other. To make use of $q_i$, Pioneer creates two additional queries, $q'_s = (q \cup q_i)\mid q_i$ with required variance $v_i$, and $q'_s = (q \cup q_i)\mid q$ with required variance $v'$, and inserts a new subset $\{q_i, q_s, q'_s\}$ into $Q$, which answers $q$ by $q'' = q - q_i$. As in the above discussions, the effectiveness of refining $q_i$ is doubled as it also improves the accuracy of $q_s$. $q_s$ cannot be refined, and its required variance $v'$ is solved for during plan creation.

Next we explain how to build $Q$ from a given set of past queries $\Phi$. Figure 13a illustrates an example with $\Phi = \{q_1, q_4\}$, with required accuracy $v_1, v_4$. Among them, $q_1$ and $q_4$ have the same required accuracy $v_1 = v_4$; further, they can share the same privacy budget, which happens, e.g., when $q_4$ is issued after $q_1$, and answered without additional privacy budget usage. Although $q_4$ is disjoint with $q$, we cannot handle it as in Figure 12a, because $q_4$ shares the privacy budget with $q_1$. Instead, Pioneer considers $q_1$ and $q_4$ as a single query that overlaps with $q$. To build $Q$, Pioneer first builds the query $q_{new}$ whose selection clause is the union of those of $q_1, q_4$, as shown in Figure 13a. In the following, we simply say that $q_{new}$ is the union of $q_1, q_4$. Then, Pioneer creates new queries $q_{s} \sim q_{new}$ (Figure 13b), with selections $q_2, q_3, q_5, q_6, q_7, q_8$, respectively, and required accuracy $v_i = v_1, v_4, v_1, v_2, v_3, v_4$, where $v'$ is the required variance for $q'$ in the execution plans of Section III. After that, Pioneer adds 3 subsets $\{q_1, q_5, q_8\}, \{q_2, q_6, q_7\}$ and $\{q_3, q_4, q_7\}$ to $Q$. Since $q_8$ is present in all three subsets, $q$ needs a composite plan.

![Figure 13: Example of building $Q$ from past queries](image)

Using the above method, for an incoming query $q$, all past queries can be used to build an execution plan for $q$. Pioneer does not take this option, because (i) it creates numerous new queries that share the same privacy budget with past ones, which render them less versatile in answering future queries, and (ii) the additional reduction in privacy budget usage for $q$ gradually diminishes with increasing number of subsets in $Q$. Instead, Pioneer chooses the $k$ (a system parameter) groups of past queries with lowest total variance to form the set $\Phi$. Each group of queries share the same privacy budget, e.g., $q_1$ and $q_4$ in Figure 13. Figure 14 shows the algorithm $CQ$ for building $Q$, for the set of queries $\Phi$. $CQ$ builds the union $q_{new}$ of all queries in $\Phi$, and creates a new query for each group, as well as $q_i$ that extends their respective unions to $q_{new}$ (lines 5-6). For instance, in Figure 13, the new query $q_7$ created for the $\{q_1, q_4\}$ is the difference between $q_{new}$ and the union of $q_1$ and $q_4$.

$CQ$ algorithm: Create $Q$

Finally, we mention that query processing in existing DP-compliant DBMSs (e.g., PINQ [16]) can be improved using ideas from this section. We call the new method Baseline, and evaluate it in our experiments. Specifically, when a new query $q$ arrives, Baseline checks whether there exists a past query $q_i$ whose selection is disjoint with that of $q$ (Figure 12a). If so, the method coalesces $q_i$ and $q$ into a single query, e.g., using the partition operator [16]. The privacy budget usage of the coalesced query is equal to that of whichever of $q_i$ and $q$ has the higher accuracy requirement. This approach consumes less privacy budget than processing $q_i$ and $q$ separately.

V. EXPERIMENTS

We implemented Pioneer, Baseline (described in Section IV), and a naive algorithm (Naive) that processes each query independently. Naive is used in most existing systems such as PINQ [16]. All methods are evaluated on a 2.5 GHz Intel Core i5 CPU with 4GBbytes of RAM. We exclude offline methods in our comparisons since they are ineffective for queries with different accuracy requirements, as explained in Section II-B. The experiments use mainly three performance metrics: total privacy budget usage for answering a sequence of queries, running time for plan and noise computation, and the number of queries each method can process under 1-DP. Properties of
DP ensure that the evaluation results only depend on the query workload, not the underlying database. To test Pioneer on real datasets, at the end of this section we also evaluate its average relative error for a series of queries, each with a given privacy budget $\epsilon$. To do this, we use Pioneer to compute the query accuracy that leads to budget usage $\epsilon$ through binary search. The real dataset used is Social Network [10], which contains the node degrees of a social network with 11,342 nodes.

As there are no standard workloads for $\epsilon$-DP, we generated simple synthetic ones, as follows. Each query is a 1D range-count, whose sensitivity is 1. The range selection is over a finite, discrete domain. The range width, start position, and required accuracy (noise variance) of each query follow Gaussian distributions, whose mean and standard deviation are parameters under investigation. We chose Gaussian since it models well many variables in practice, e.g., the age of patients suffering from a certain disease. Further, a high standard deviation (STD) approximates uniform distribution, whereas a low STD models high skewness. We investigate the impact of 7 workload parameters: number of queries $N$, size of the queried domain $D$, the mean $l_{avg}$ and STD $l_{std}$ of the query range width, the mean $v_{avg}$ and STD $v_{std}$ of the required noise variance for each query, and the STD $s_{std}$ of the range start position. We also evaluate the impact of parameter $k$, which determines the number of past queries used in a plan (Section IV). Table III lists the parameter values, with defaults in bold.

### Table III

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range and (in bold) Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>80, 400, 2000, 10000, 50000</td>
</tr>
<tr>
<td>$D$</td>
<td>500, 1000, 2000, 5000, 10000, 20000</td>
</tr>
<tr>
<td>$l_{avg}$</td>
<td>20, 80, 320, 1280, 5120</td>
</tr>
<tr>
<td>$l_{std}$</td>
<td>2, 10, 50, 250, 1250</td>
</tr>
<tr>
<td>$v_{avg}$</td>
<td>10000, 50000, 250000, 1250000, 6250000</td>
</tr>
<tr>
<td>$v_{std}$</td>
<td>1000, 5000, 25000, 100000, 500000</td>
</tr>
<tr>
<td>$s_{std}$</td>
<td>10, 20, 50, 100, 200</td>
</tr>
<tr>
<td>$k$</td>
<td>0, 1, 2, 3, 4, 8, 32, 64, 256</td>
</tr>
</tbody>
</table>

In each experiment, we vary one parameter, and fix the rest to their defaults. We ran each configuration 10 times, and report the average results. We use relatively large noise variance (e.g., 250,000, corresponding to standard deviation 500), which may suit applications with relatively large values for the actual results (e.g., population counts). Nevertheless, the magnitude of the noise variance does not affect the relative performance of the methods, or the trends in their performance with varying parameters. For instance, if we decrease $v_{avg}$ from 250,000 to 2,500, the privacy budget cost of each method will simply multiply by a factor of 10 in each setting.

Figure 15a plots the total privacy budget consumption as a function of the workload size $N$. Both axes are in logarithmic scale. The total budgets of all methods increase with $N$, with Pioneer consistently outperforming the other two, often by more than an order of magnitude for a relatively large workload. Meanwhile, the budget consumption of Naive and Baseline grow linearly with $N$, whereas the privacy cost of Pioneer increases at a much slower pace and eventually flattens. This is because the more queries are processed, the higher the chance for Pioneer to identify past queries that answer an incoming query with sufficient accuracy, leading to little budget usage. Figure 15b shows the CPU time for Pioneer and Baseline for the entire workload. The CPU time for Naive is omitted, as it lies flat along the X axis. Pioneer needs no more than a few milliseconds to optimize a query, which is negligible, especially considering its vast reduction in privacy budget consumption. Hence, in the following experiments, we omit CPU time results.

Figure 16a varies the size $D$ of the domain on which the range selections are performed. The results suggest that $D$ does not have a significant impact on privacy cost. This is true even for very small domains (e.g., 500), where the query (with default average length 320) covers a large portion of the domain. Hence, unlike query optimization in traditional DBMSs, selectivity may not be a critical factor in our application. Similarly, Figure 16b investigates the effect of another parameter that affects selectivity, namely the average width $l_{avg}$ of the query range. The performance of all three methods remains stable, except for very small values of $l_{avg}$ e.g., 20, which are comparable to the standard deviation $l_{std}$ of the query width. Such a parameter combination generates queries with tiny ranges, which tend to be disjoint, and, thus, can often be coalesced by Baseline and Pioneer, leading to lower privacy costs. The privacy budget usage of Naive however is unaffected by $l_{avg}$ since it processes each query independently.
Figure 16c compares the methods for different values of the standard deviation of the range width \( l_{std} \). Naive and Baseline are not sensitive to \( l_{std} \) but the privacy budget usage of Pioneer grows with \( l_{std} \). The reason is that with small values of \( l_{std} \), queries tend to be rather similar, sometimes even identical. For these settings, LR plans and hybrid/composite plans with lots of query refinement perform quite well, since they often lead to a small number of repeatedly refined queries with very high accuracy. The combinations of these queries often answer a considerable fraction of new queries without using additional privacy budget, which reduces total privacy costs. As \( l_{std} \) grows, this phenomenon gradually diminishes, and the performance of Pioneer stabilizes. Nevertheless, in terms of privacy budget usage, Pioneer consistently outperforms the other two methods, even for large values of \( l_{std} \). Figure 16d shows the effect of the standard deviation of the start position for query ranges \( s_{std} \). The mean value for query start positions does not affect our evaluation results. Pioneer is again the winner in all settings. As \( s_{std} \) grows, the privacy costs of Pioneer increase at first, reaching a peak at around \( s_{std}=50 \), and drop slightly after that. The initial privacy cost increase is due to the fact that queries become more diverse, reducing the effectiveness of LR-like plans, as explained above. The subsequent drop is caused by more disjoint queries being coalesced together, which also benefits the Baseline method.

Next we focus on the impact of the required query accuracy. Figure 17a shows the results with different average required variance \( v_{avg} \). As expected, the total privacy budget usage decreases with growing \( v_{avg} \), and the trend agrees with Equation (3). The standard deviation \( v_{std} \) of required noise variance has a subtle effect, as shown in Figure 17b: when \( v_{std} \) becomes high (e.g., when \( v_{std}=\text{avg}=250,000 \)), the budget consumption slightly decreases. This is because such settings generate queries with very different accuracy requirements, and the most accurate ones can often be composed to answer the most inaccurate ones with little or no budget consumption.

To demonstrate the practical utility of Pioneer, we repeated all experiments, and report the number of queries answered by a DP-compliant DBMS with a total privacy budget \( \epsilon = 1 \), which is a popular value for \( \epsilon \). The maximum number of queries is bounded by the size of the workload \( N \), which is 10,000 by default. Figure 18 shows the results of varying parameters \( l_{avg}, l_{std}, v_{avg} \), and \( v_{std} \). The comparisons of Pioneer, Baseline and Naive lead to the same conclusions as before. In Figure 18d, the effect of varying \( v_{std} \) on Pioneer can be more clearly observed. In all settings, Pioneer handles 5-10 times as many queries as Baseline and Naive.

Having established the superiority of Pioneer over the other methods, we next investigate the intrinsic properties of Pioneer. Figure 19 shows the effect of parameter \( k \), i.e., the number of (possibly coalesced) previous queries used to answer new query \( q \). Since there is no competitor, we show the impact of \( k \) with different values of \( l_{std} \) (standard deviation of range width) and \( s_{std} \) (standard deviation of query start position). For all tested values of \( l_{std} \) and \( s_{std} \), Pioneer’s privacy budget usage decreases quickly as \( k \) grows, until reaching a stable point at around \( k=4 \). After that, further increases in \( k \) no longer have a significant impact. The reason for this behavior is that larger \( k \) values cause more previous queries to be involved in the plan. The initial few chosen by our query selection algorithm are those that represent the new query \( q \) most accurately, leading to a sharp privacy cost decline. After that, adding more queries with lower accuracy does not help much, according to Equation (15). Meanwhile, a high value for \( k \) leads to a complicated plan, which incurs higher computational costs. Overall, values of \( k \) from 4 to 10 strike a good balance.

Finally, Figure 20 evaluates the methods on a real dataset Social Networks, and report the average relative error, i.e., the ratio between noise and exact query answer, for a query sequence where each query has a given privacy budget \( \epsilon = 0.001 \). The main difference from the results above is that certain parameters (e.g., query length) affect the exact query
answer, and, thus, the relative error. Nevertheless, Pioneer is again the clear winner in all settings.

Summarizing the experiments, Pioneer outperforms Baseline and Naive in all settings in terms of both privacy budget consumption, and the maximum number of queries that can be processed under 1-DP. There is a considerable performance gap between Pioneer and the other two, often over an order of magnitude. CPU costs for plan selection are negligible in most settings. Thus, we believe that Pioneer is the method of choice for query optimization in a DP-compliant DBMS.

VI. CONCLUSION

This paper proposed Pioneer, an effective and efficient query optimizer for DP-compliant DBMSs. Pioneer builds an execution plan that reuses past queries, in an effort to minimize privacy budget usage. It also includes an effective algorithm for selecting past queries that contribute to answering the incoming one, and both plan computation and past query selection incur negligible computational costs. Extensive experiments show that for a given privacy budget, Pioneer can process 5-10 times as many range-count queries as the naive solution commonly used in existing systems. For a fixed workload of range queries, Pioneer typically consumes an order of magnitude less privacy budget than existing methods. The current version of Pioneer focuses on minimizing the privacy budget usage only for the current query $q$ under processing. Hence, one option for future work is to select the query plan based on both the privacy cost of $q$, and the plan’s expected impact for future queries.

ACKNOWLEDGEMENT

Shangfu Peng did his work while doing an internship at ADSC. This work is by Singapore’s A*STAR, under SERC Grant 102-158-0074.

VII. REFERENCES