On Privacy of Multidimensional Data Against Aggregate Knowledge Attacks

Ala Eddine Laouir(✉) and Abdessamad Imine

Lorraine University, Cnrs, Inria, 54506 Vandœuvre-lès-Nancy, France
{ala-eddine.laouir,abdessamad.imine}@loria.fr

Abstract. In this paper, we explore the privacy problem of individuals in publishing data cubes using `SUM` queries, where a malicious user is expected to have an aggregate knowledge (e.g., average information) over the data ranges. We propose an efficient solution that maximizes the utility of `SUM` queries while mitigating inference attacks from aggregate knowledge. Our solution combines cube compression (i.e., suppression of data cells) and data perturbation. First, we give a formal statement for the privacy of aggregate knowledge based on data suppression. Next, we develop a Linear Programming (LP) model to determine the number of data cells to be removed and a heuristic method to effectively suppress data cells. To overcome the limitation of data suppression, we complement it with suitable data perturbation. Through empirical evaluation on benchmark data cubes, we show that our solution gives best performance in terms of utility and privacy.

Keywords: Data cubes · Privacy preservation · Cell suppression · Cell perturbation · Cube compression

1 Introduction

Multidimensional data (or data cubes) are widely used in many fields to store all collected data, as these data structures are optimized for Online Analytical Processing (or OLAP) [2,8,9]. For business or research purposes, the data collected is made available to external parties (e.g., analysts, and organizations) to enable them to query and analyze trends and patterns necessary for decision making. Although most external parties have legitimate usage interests and data is anonymized before publication or query, there are situations where a malicious user can mine this data in order to endanger the privacy of individuals, such as leaking medical records. Because of this privacy risk, many research works have addressed this issue and different models have been proposed [5,12]. For example, aggregation of a single cell value is not allowed, and query set size and access controls are deployed to provide additional security (for more details see [15]). However, most of the proposed techniques focus on the privacy of individual data (or cells of data cubes) and neglect insights that can be gained by simply

This work is funded by DigiTrust (http://lue.univ-lorraine.fr/fr/article/digitrust/).

J. Domingo-Ferrer and M. Laurent (Eds.): PSD 2022, LNCS 13463, pp. 92–104, 2022.
https://doi.org/10.1007/978-3-031-13945-1_7

analyzing aggregate patterns (such as average information) [3]. These aggregate patterns may not be allowed for security reasons, but they can be easily inferred in many cases. Even for multidimensional data where only `SUM` is allowed, knowledge of the average could be a valuable indicator to know the trend of the data (e.g., whether or not the average salary is above the minimum wage).

Illustrative Example. Consider that a large retail company allows access to sales from all its stores during the year through range sum queries (here and in the rest of the paper, we consider a data cube that contains only positive values). Due to business and analytical needs, our retail company publishes a view (or part) of the data cube and makes it accessible only through `SUM` range queries (see Fig. 1). Even though the `COUNT` information is not explicitly available, a malicious user can get it easily: either by knowing some metadata of the published cube and queries [3], or by using sophisticated approaches such as the Volume Leakage attack [6]. The result of the aggregate average `AVG` can then be inferred and exploited to violate the privacy of individual cells.

Suppose our malicious user knows that two stores had similar sales on certain days (weekend, holidays, events, etc.). Using the result of `SUM` query and the knowledge of `COUNT`, he can deduce the result of `AVG`. For the region defined by the range $\{(Shop2 : Shop3), (Day2 : Day3)\}$ in Fig. 1, the `AVG` is equal to $129.5k$. Based on this `AVG`, the attacker can now estimate the sum of the range $\{(Shop2 : Shop4), (Day2 : Day2)\}$ as $(129.5 + 129.5 + X)$ and deduce that X is $172k$, which gives the cell value with high precision. The attack can proceed in the same way to disclose the rest of the values. Note that the attacker can also assume that the negative or small scale results are the empty cells, allowing him to reconstruct the entire region.

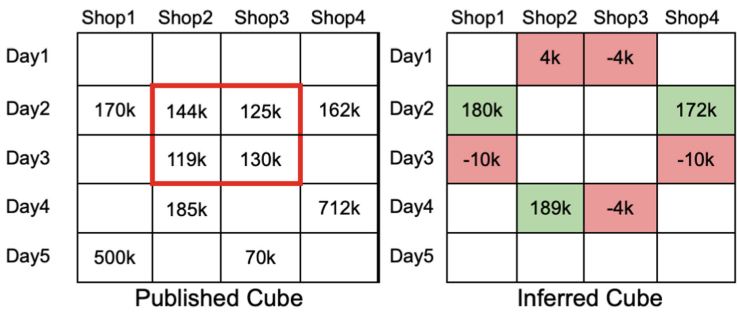

Fig. 1. Data cube for retail company.

Contributions. To disturb the `AVG` and prevent these attacks, cell suppression techniques are better suited since they simultaneously modify the `COUNT` (by -1) and the `SUM` (by *minus* the value of the cell), for each cell suppressed [3]. Cell suppression is very similar to a well-known technique for speeding up query responses in OLAP, namely *cube compression*.

In this paper, we present an efficient method that maximizes the utility of `SUM` queries while mitigating inference attacks from `AVG` aggregate knowledge.

Our method combines cube compression via cell suppression and cell perturbation to ensure better utility and privacy. Our contributions are as follows:

1. We develop a Linear Programming (LP) model for finding the optimal allocation needed to provide the best utility/privacy results.
2. We design a cell suppression technique that maximizes privacy against attacks based on `AVG`, and can be extended to other aggregate operators but this will be the subject of future work.
3. To overcome the limitation of cell suppression, in the case where the values are close to each other, we complement it with suitable cell perturbation to ensure a higher level of privacy while maintaining utility.
4. All of our contributions are validated by extensive experiments, in which our techniques have outperformed the state-of-the-art [3] approach by ensuring both utility and privacy against attacks that use `AVG` aggregate inferred from data cubes.

Paper Organization. In Sect. 2, we review related work. In Sect. 3, we introduce the notation used and the problem statement of the aggregate knowledge privacy based on data suppression. We present our privacy-preserving method in Sect. 4. We provide an experimental evaluation of our method in Sect. 5 and discuss the limitations of our solution in Sect. 6. Finally, we conclude in Section 7 by presenting some future works.

2 Related Work

Most of the privacy techniques for OLAP data are derived from the literature on the privacy of statistical database [11]. These techniques can be classified into two categories: Access restriction/control methods and disruption methods.

Restriction and Access Control Methods. In [13], they presented two types of inference attacks and then proposed an access control system that further restricts the privileges of each user until they become inference free. In [16], the attacker can use the knowledge about the cardinality of the empty cells, combined with the `SUM` singleton queries, to infer the individual values. Then, [16] proposed a privacy method by dividing the cube into blocks and only keeping the blocks that are not compromised. [14] is an enhancement of [16] with an audit control system allowing only range queries that are inference-free. [17] is another query auditing method that uses information theory and only responds to user queries if the user's prior knowledge (represented by his previous queries) and new knowledge do not compromise the cells targeted by the query.

Perturbation Methods. Another way to provide privacy is to add noise to the cube cells, in such a way any inference of a cell's value will yield a perturbed value and reconstruction will not give the original data. [1] presented an approach to add noise to cells, where each cell is kept as it is with probability p or noised (using a noise sampled from the normal distribution) with probability $1 - p$.

In [11], they present another method where the cube is considered as a group of blocks that will be perturbed individually. In each block, the sum of the noise added to each cell in a row (as well as for the cells in a column) is equal to 0 so that it can provide accurate range sum queries. Their results show that they were able to change the values significantly, while providing accurate answers to queries. In [3], they present another perturbation approach that uses cell deletion (also called sampling). Based on the average aggregate knowledge, their algorithm applies data suppression to modify not only the response of the queries but also the aggregate patterns inferred from the queries. They compared to [11], and the results show that adding noise alone cannot prevent this type of inference without the loss of utility and cell suppression is a better suited solution. Our work targets the same aspects of inference and privacy as defined in [3], and we have provided a better algorithm for cell suppression and perturbation.

Another perturbation method is the Differential Privacy (DP) [5], considered the gold standard. However, DP cannot be applied to all the possible scenarios [4]. In our work, we considered `SUM` queries, and one of the main problems in applying DP is to define the global sensitivity for the `SUM` functions [10]. Also it requires the addition of a lot of noise to significantly disrupt the `AVG`, resulting in poor utility. For these reasons, DP is out of the scope of this work and given other privacy considerations, we will investigate the application of differential privacy to secure high-dimensional data (multidimensional cubes) in future work. Another privacy model is k-anonymity [12], which was used by [8] to avoid re-identification attacks on the dimensions of a data cube. This is also outside the scope of this paper.

3 Problem Statement

In this section, we give the notation used throughout the paper and a formulation of the problem under consideration.

3.1 Preliminaries

Data Cube. A *data cube* C is a multidimensional data over a relational table for a set of *dimensions* $D = \{d_1, d_2, \ldots, d_n\}$, where each dimension d_i corresponds to an attribute and each cell contains the result of an aggregated *measure*. Figure 1 illustrates a 2-dimensional cube where attributes *Shops* and *Days* are dimensions, and each cell contains the *measure* which is the result of the aggregate operator `SUM` on the amount of sales. In this work, we consider only `SUM`, as this aggregate operator is (i) extensively used in many multi-dimensional frameworks [15], and (ii) used to compute other aggregate operators such as `AVG`.

We consider here a special class of queries called *continuous range queries* in such a way that `SUM` is performed over a range query $R = \{r_1, r_2, \ldots, r_n\}$, noted by $\texttt{SUM}(R)$, where r_i is a continuous range on dimension d_i specified by the start and end positions. For instance, in Fig. 1, $\texttt{SUM}(R)$ results in $1004k$ with $R = \{(Shop1 : Shop3), (Day3 : Day5)\}$. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ be a *query workload* used by the data publisher and the end user as a contract that defines

the view to be published (via *range queries*) instead of the whole data cube. Let $|R|$ be the size (i.e., the number of non-empty cells) of range query R. In Fig. 1, the size of $R = \{(Shop1 : Shop3), (Day3 : Day5)\}$ is 5.

Metrics. Our goal is to suppress (and possibly perturb) some cell values from each range R to get R'. Any privacy-preserving data publishing method for data cubes is evaluated on two criterias: *utility* and *privacy*. As utility objective, we consider boosting the accuracy of `SUM` queries in our solution. For that, we compute the *relative error of accuracy* $\mathcal{A}_e$ which shows how different the `SUM` answer on the exact range R and the altered one R':

$$\mathcal{A}_e(R, R') = \frac{|\texttt{SUM}(R) - \texttt{SUM}(R')|}{\texttt{SUM}(R)} \tag{1}$$

It is clear that the smaller the relative error of accuracy, the better the utility.

As for privacy issues, our objective is to prevent inference attacks like those presented in Sect. 1. Let us recall that in these attacks the information on the average was deduced from the data cube. Therefore, we compute the *inference error* $\mathcal{I}_e$ which shows how well the solution we propose is able to disrupt this inferred average and thus mitigate successful attacks [3,7]:

$$\mathcal{I}_e(R, R') = \frac{|\texttt{AVG}(R) - \texttt{AVG}(R')|}{\texttt{AVG}(R)} \tag{2}$$

Note that the higher the inference error, the better the privacy of the data cube. Given both metrics, a good privacy-preserving data publishing method should provide the best utility-privacy tradeoff.

3.2 Problem Definition

Let C be a data cube with a query workload $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$. We consider attackers whose knowledge is limited to the average information and/or the distribution of some cells (as described in Sect. 1). The attackers can discover the exact and/or approximate values of other cells, and accordingly infer sensitive attribute information. To prevent these attacks, we propose to suppress some values from data cube C (only the region defined by $\mathcal{R}$) while preserving the accuracy of `SUM` queries and mitigating the inference due to the use of the average operator. More precisely, for each R_i in $\mathcal{R}$, we create a non empty replica R'_i that contains a *minimal subset* of the cells in R_i and the others are left null. Our privacy-preserving solution can be defined as a multi-objective optimization problem:

$$\begin{aligned}
&\textbf{maximize} \ \textstyle\sum_{k=1}^{m} |R_i| - |R'_i| \\
&\textbf{minimize} \ \tfrac{1}{m} \textstyle\sum_{k=1}^{m} \mathcal{A}_e(R_i, R'_i) \\
&\textbf{maximize} \ \tfrac{1}{m} \textstyle\sum_{k=1}^{m} \mathcal{I}_e(R_i, R'_i) \\
&\textbf{subject to} \ R'_i \subset R_i \text{ for each } R_i \in \mathcal{R}
\end{aligned} \tag{3}$$

The first objective function of Eq. 3 maximizes the difference between R_i and R'_i to meet the requirement of cube compression. In real world scenarios, the sizes of data cube and its query workload are very huge. Finding an optimum solution to multi-optimization problem 3 is hard and intractable task. This type of resolution falls under the broad category of multi-criteria, or vector optimization problems. Unlike single-objective problems, it is not always possible to find an optimal solution that satisfies all the function objectives under consideration. Moreover, even several solutions may not meet the expectations of the data publisher in terms of utility and privacy requirements. Therefore, it would be more beneficial if the data publisher had the ability to select and verify their own levels of utility and privacy.

In the next section, we propose a heuristic to problem 3 combining cell suppression and perturbation and allowing us to find plausible solutions.

4 Privacy-Preserving Method

We present a method to publish a view of a given data cube while ensuring good utility of `SUM` queries and protecting cells privacy against average-based inference attacks. The view is built using two operations on non-empty cells: suppression and perturbation of data cells. Our method proceeds in three steps: (i) splitting query ranges at finer grid granularity in order to increase the accuracy of potential `SUM` queries; (ii) defining and solving a Linear Programming (LP) problem to determine how many cells are needed from $R_i \in \mathcal{R}$ and its sub-ranges to prepare a range view given a storage space $\mathcal{B}$; (iii) preparing range views by deleting some data cells and perturbing others as necessary as possible.

4.1 Preprocessing Step

Each R_i in the query workload $\mathcal{R}$ represents a region of the data cube that we want to publish while ensuring utility and privacy. If the range of R is large, then dealing directly with this range would be a coarse granularity to the problem, thereby penalizing small user queries after the data cube view is published. Indeed these queries will not be efficient (i.e., null result) since they target regions where cells have been deleted unnecessarily due to coarse granularity. Accordingly, to ensure that the retained/removed cells from region R_i are evenly distributed across all parts, we consider a smaller unit on which to apply cell suppression. We call this unit a sub-query (or sub-region), so each R_i will be divided (logically by indices only) into smaller sub-regions. We build a view for each sub-region of R_i, which together constitute the global view of R_i. Splitting data into smaller units is a common preprocessing step, either on the data cube directly [11] or on a query workload [3].

4.2 Space Allocation Step

Unlike multi-objective optimization problem 3 (where the decision variables are the cells values of each R_i), we define a simple LP problem to compute the

view size of ranges as well as their sub-ranges (resulting from the preprocessing step). Given an allocated space $\mathcal{B}$ in which we want to build a view of the query workload $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$. The objective is to distribute this space in such a way to guarantee a minimum $\mathcal{A}_e$ for all R_i:

$$\textbf{minimize } \frac{1}{m} \sum_{i=1}^{m} \frac{|\texttt{SUM}(R_i) - \texttt{SUM}(R'_i)|}{\texttt{SUM}(R_i)} \tag{4}$$

The view R'_i is a subset of R_i to be published. The size and the data cells of R'_i are unknown at this stage. As the data cells of R_i are known, we can compute the average value $\texttt{AVG}(R_i) = \texttt{SUM}(R_i)/|R_i|$. Instead of determining which data cells of R_i to include in R'_i, we consider a single a decision variable b_i, the size of R'_i, and replace the data cells of R'_i by a single cell $\texttt{AVG}(R_i)$. Thus, Eq. 4 is redefined as follows:

$$\textbf{minimize } \frac{1}{m} \sum_{i=1}^{m} \frac{|\texttt{SUM}(R_i) - b_i \times \texttt{AVG}(R_i)|}{\texttt{SUM}(R_i)} = |1 - \sum_{i=1}^{m} \frac{1}{m \times |R_i|} \times b_i| \tag{5}$$

From Eq. 5, our allocation problem needs only the size of each R_i in $\mathcal{R}$. Therefore, our LP problem is stated as follows:

$$\begin{aligned} &\textbf{minimize } \ |1 - \textstyle\sum_{i=1}^{m} \frac{1}{m \times |R_i|} \times b_i| \\ &\textbf{subject to } \textstyle\sum_{i=1}^{m} \mathtt{b}_i <= \mathcal{B} \\ &\qquad\qquad \mathtt{minimum}_{space} <= \mathtt{b}_i < |R_i| \text{ for each } i \in \{1, \ldots, m\} \end{aligned} \tag{6}$$

The first constraint of Eq. 6 ensures that the space allocated to all range queries does not exceed the given space $\mathcal{B}$. As for $\mathtt{minimum}_{space}$, it is an input parameter passed to the algorithm to ensure that each query gets a minimum space allocation. The solution of Eq. 6 will give us the allocated space b_i for each R_i, and since each R_i consists of a group of subqueries, we need to divide the b_i and give each subquery its appropriate allocation. To do this, we reuse our LP problem to calculate the space allocation for each subquery of R_i but with $\mathcal{B}$ equal to the space found for R_i.

4.3 View Creation Step

After the allocation step, each query (and its subqueries) in $\mathcal{R}$ would be allocated a space (in number of cells). To prepare the data cube view for publishing, the next step is to find the cells to keep in each region R (working on each of its subregions) relative to $\mathcal{A}_e$ and $\mathcal{I}_e$. To this end, we have designed two algorithms, the first based solely on cell suppression and the second being a perturbation-based approach.

Cell Suppression Algorithm. To get the optimal $\mathcal{A}_e$ and $\mathcal{I}_e$, we have to try all possible combinations of cells (that fit in the allocated space b), which is inconvenient and computationally expensive. So an approximate solution might

be a better way to solve this problem. In [3], they used a heuristic that considers first the utility (by selecting outliers or the largest values), and privacy in second order.

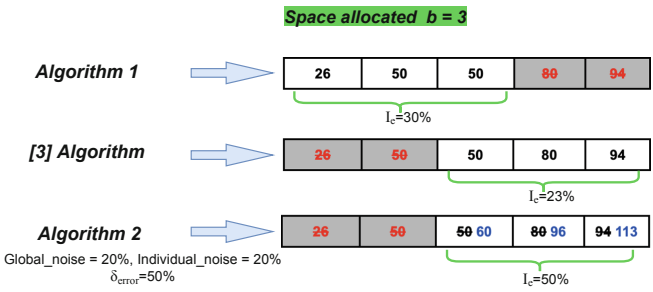

Fig. 2. Output comparison between Algorithm 1, Algorithm 2 and [3].

Algorithm 1. Cells suppression

Inputs: r - subquery region
b - allocated space
Output: r' - view of r

$r' \leftarrow []$
$T \leftarrow$ sort in asc. order r by values
$(index, best\_index, best\_\mathcal{I}_e) \leftarrow (0, 0, 0)$
while $index + b < |r|$ **do**
  $r' \leftarrow$ get b values of T from position $index$
  if $\mathcal{I}_e(r, r') >= best\_\mathcal{I}_e$ **then**
    $best\_\mathcal{I}_e \leftarrow \mathcal{I}_e(r, r')$
    $best\_index \leftarrow index$
  end if
  $index \leftarrow index + 1$
  $r' \leftarrow []$
end while
$r' \leftarrow$ get b values of T from position $best\_index$
return r'

Algorithm 2. Cell perturbation

Inputs: r - subquery region
b - allocated space
$noise_{global}$ - maximum distortion
$noise_{individual}$ - distortion rate
δ_{error} - reachable level of $\mathcal{I}_e$
Output: $perturbed\_r'$ - noised view of r

$r' \leftarrow$ get b largest values from r
$noise\_budget \leftarrow \text{SUM}(r) \times (1 + noise_{global}) - \text{SUM}(r')$
$noise\_step \leftarrow 1$
$perturbed\_r' \leftarrow r'$
$noise \leftarrow 0$
while $(\mathcal{I}_e(r, perturbed\_r') < \delta_{error})$ **and** $(noise < noise\_budget)$ **and** $(noise\_step < noise_{individual})$ **do**
  $perturbed\_r', noise \leftarrow add\_noise(r', noise\_step)$
  if $noise > noise\_budget$ **then**
    $perturbed\_r', noise \leftarrow add\_noise(r', noise\_step - 1)$
    return $perturbed\_r'$
  end if
  $noise\_step \leftarrow noise\_step + 1$
end while
$perturbed\_r', noise \leftarrow add\_noise(r', noise\_step)$
return $perturbed\_r'$

Relegating $\mathcal{I}_e$ to the second order of priority does not guarantee the best results in all cases, so we propose Algorithm 1 that optimizes $\mathcal{I}_e$ in priority.

Given a sub-region r of a region R and space allocation b, the algorithm first sorts the cells by values (each cell contains a value and a location in the cube) in ascending order. It computes iteratively the maximal value of $\mathcal{I}_e$ by considering successive and overlapping b values. At the end, Algorithm 1 constructs r' with the subset of cells that gave the best results (R' is composed of all r'). Figure 2 shows by an example the difference between the results provided by our heuristic and [3]. In our solution, we give priority to $\mathcal{I}_e$, but without neglecting $\mathcal{A}_e$. If the first and last b values yield the best (and similar) $\mathcal{I}_e$, our solution will choose the last b values. Since the values are ordered, this set of cells offers the least (best) $\mathcal{A}_e$.

Cell Perturbation Algorithm. When the values are close to each other, cell suppression alone will not be able to provide good $\mathcal{I}_e$. For these cases, we have

designed Algorithm 2, which is based on cell suppression and perturbation and can be triggered when Algorithm 1 does not perform well. In addition to r and b, it takes three other parameters: δ_{error} represents the level of $\mathcal{I}_e$ we want to reach, $noise_{individual}$ limits the noise (distortion rate) added to each cell individually, and $noise_{global}$ defines the maximum distortion rate between r and r' (i.e., $\mathcal{A}_e$) allowed by adding noise. These additional thresholds ensure that the added noise does not affect the utility (i.e., $\mathcal{A}_e$) more than necessary, and also control the distortion applied to each cell. The algorithm first chooses the cells with the largest values to build r', then calculates the $noise_{budget}$ based on $noise_{global}$ and the noise generated by suppressing the cells (discarding the most small values). A noise is added to the cells of r' (the function *add_noise* returns a perturbed version of r' and the amount of noise added) incrementally using *noise_step* which represents the distortion rate (e.g. in Algorithm 2 it is equal to 1%). This process is repeated each time with bigger *noise_step* as long as: (i) $noise_{global}$ is not exceeded (ii) δ_{error} is not satisfied, and (iii) *noise_step* is smaller than $noise_{individual}$. Figure 2 shows that the perturbation method provides better utility and privacy than other suppression-based algorithms.

Using these different thresholds allows us to better control the balance between utility and privacy. For example, by allowing more noise (accuracy limit defined by $noise_{global}$), we can ensure a higher δ_{error}.

5 Experimental Evaluation

For experimental evaluation, we implemented[1] our method and [3] approach (as this one is closest to our work) in C# to facilitate compatibility with SSAS[2] used to create data cubes from TPCDS[3] and AdventureWork2012[4] (see Table 1 for the details on the cubes). For solving our LP problem in allocation step, we used Google OrTools with SCIP solver[5]. To measure the performance of our method, we conducted several experiments to observe how $\mathcal{A}_e$ and $\mathcal{I}_e$ vary depending on two parameters. The first parameter is the allocated space $\mathcal{B}$ which we varied relatively (e.g., 50%) from the original size of $\mathcal{R}$ in the data cubes in order to check the performance of $\mathcal{A}_e$ and $\mathcal{I}_e$ during the allocation step. The second parameter is the selectivity of each range query $R = \{r_1, r_2, \ldots, r_n\}$ defined as $\|R\| = (|r_1| * |r_2| * ... * |r_n|)$ where $|r_i|$ represents the length of the range on the i_{th} dimension. In the experiments, we create a random query workload $\mathcal{R}$, with initial selectivity for all queries. Next, we modify (multiply by 2, 3, ...) the initial selectivity to observe how the performance changes if the size of the input data increases (with $\mathcal{B}$ between 50% and 60%). Our experiments are divided into two parts: the first part compares our method to [3] in terms of cell suppression and

[1] https://github.com/AlaEddineLaouir/PUV-CUBE.
[2] https://docs.microsoft.com/fr-fr/analysis-services/?view=asallproducts-allversions.
[3] https://www.tpc.org/tpcds/.
[4] https://github.com/Microsoft/sql-server-samples/releases/download/adventureworks/AdventureWorks2012.bak.
[5] https://developers.google.com/optimization.

Table 1. Data cubes used in our experiments.

Databases	Dimensions					Measures	Total
TPCDS	Product 18000	Household 7200	Store 6	Date 1824	Promo 300	Net paid Type reel	∼4.25 E14 cells
Adventure works	Product 158	Date 1124	Customer 18484			InternetSales Type reel	∼36 M cells

space allocation. The second part is devoted to our method in order to compare the techniques of suppression and perturbation of cells.

In the first part, we used the TPCDS data cube to compare both methods in terms of space allocation using the cell suppression technique of [3] (**Experiment (A)**) and our cell suppression technique (**Experiment (B)**). In Fig. 3, we find the expected behavior of our method in both experiments **(A)** and **(B)**. Indeed, both $\mathcal{A}_e$ and $\mathcal{I}_e$ converge to 0 when the given allocation is too large. This is something that [3] cannot replicate, because their allocation scheme is based on the data distribution (i.e., it relies heavily on variance analysis). If the data in the query workload is not balanced according, their allocation scheme fails to distribute the allocated space. Unlike our method, [3] gives more space than necessary to some regions and neglects others (for example, this is visible when the allocation is given at 90%), [3] still fails to get a $\mathcal{A}_e$ minimal. As for selectivity in both experiments **(A)** and **(B)**, Fig. 3 shows that our method is able to provide better results in $\mathcal{A}_e$ and $\mathcal{I}_e$ when the query size increases. Despite an evolution of the results in the right way for the experiment **(A)**, [3] nevertheless presents worse results in the experiment **(B)** when the size of the queries increases. From Fig. 3, we can also see that our view creation step provides better results than the suppression heuristic of [3] as shown in experiments **(A)** and **(B)**. From this first part of the experiments, we conclude that our method (allocation and view creation steps) outperforms [3].

For the second part of our experiments, we seek to compare the effectiveness of the perturbation algorithm Algorithm 2 and the cell suppression algorithm Algorithm 1. In addition to TPCDS, we used the Adventure Work data cube because it contains cell values that are close to each other, which allows us to better highlight the performance of both algorithms. To perform these experiments, we used Algorithm 2 with the following parameters: $noise_{global} = 20\%$, $noise_{individual} = 27\%$, $\delta_{error} = 50\%$. Applying both algorithms to the TPCDS data cube, the results for $\mathcal{A}_e$ and $\mathcal{I}_e$ are similar in all cases as illustrated in Fig. 4. Whether by allocation or selectivity, our perturbation technique did not add significant noise. On the other hand, in the case of the adventure work data cube, we find that cell suppression (see Algorithm 1) alone does not provide any level of privacy. This is due to the closeness of the cell values giving poor results for $\mathcal{I}_e$ as explained in Sect. 4.3. However, we find that the perturbation technique (see Algorithm 2) is far superior in terms of performance for $\mathcal{A}_e$ and $\mathcal{I}_e$. From the allocation, the perturbation provides less $\mathcal{A}_e$ with much more $\mathcal{I}_e$. We notice that Algorithm 1 is only able to provide 3% of $\mathcal{I}_e$, while Algorithm 2 is able to

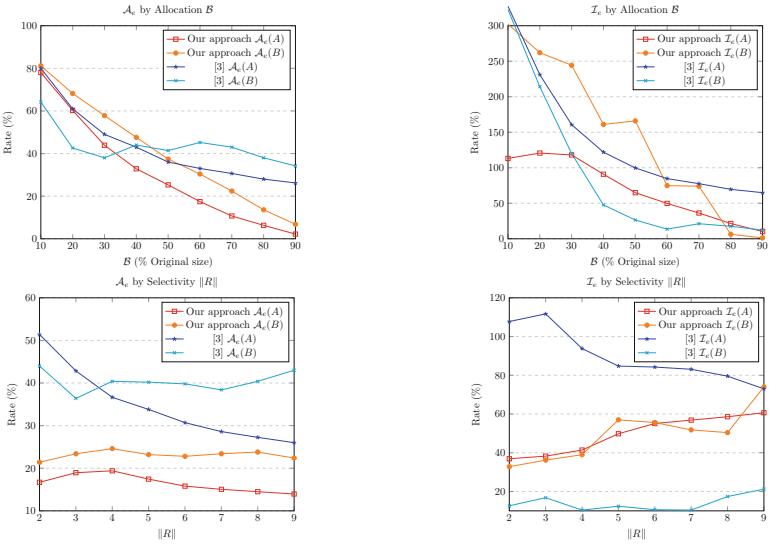

Fig. 3. Comparaison with [3] in terms of selectivity and allocation.

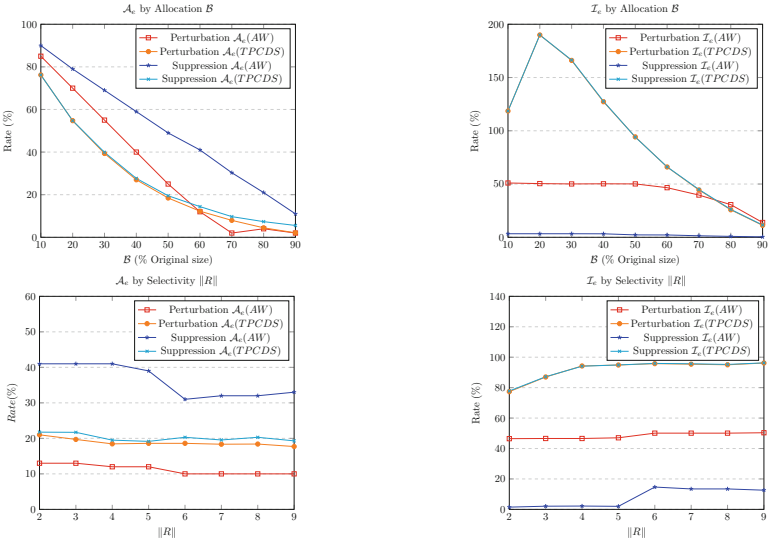

Fig. 4. Comparaison between suppression and perturbation

achieve the desired 50% of $\mathcal{I}_e$. Indeed, Algorithm 2 guarantees a better value of $\mathcal{A}_e$ because it adds noise to cells kept in view, which reduces the effect of deleted cells on $\mathcal{A}_e$. Since cell perturbation is limited by the $noise_{global}$ parameter, it cannot increase the value of $\mathcal{A}_e$ beyond what is allowed. The results also show that Algorithm 2 is able to provide the required δ_{error} passed to the parameters

and respects the utility constraint. To sum up, Algorithm 1 may be enough to provide a good balance between Utility/Privacy when the data distribution is uniform. Otherwise, in the case of close values, Algorithm 2 is suitable and gives better results.

6 Discussion

Our method incurs some computational costs and presents some limitations. Despite a cost due to the resolution of our LP problem (see Eq. 6), it should be noted that all computations are performed in offline mode and without impact on the end user. However, comparing our method and [3] in term of computation time, we can see that our allocation algorithm is faster in all experiments because it only requires the count ($|R|$) of each region compared to multiple scans of cells needed for [3]. For view creation, the cell deletion algorithm proposed by [3] takes the least computation time compared to Algorithm 1 and Algorithm 2, due to the fact that their algorithm does not test many subsets to choose the best one for privacy.

We have proposed a LP problem to divide a given space $\mathcal{B}$ over the query workload $\mathcal{R}$ to have minimal utility loss, but we can transform it into a multi-objective optimization problem by including the objective function minimizing $\mathcal{B}$. As said in Sect. 3.2, finding an optimum solution to this kind of optimization problem is hard and intractable task. In addition, to minimize the number of decision variables, we replaced the data cells of the view R'_i by the average value $\texttt{AVG}(R_i)$ where R_i is the original range (see Eq. 6). Using this simplification, it will eliminate the maximization of the inference error $\mathcal{I}_e$ (see Eq. 2) as $\texttt{AVG}(R'_i)$ will be replaced by $\texttt{AVG}(R_i)$. For this reason, we considered privacy only in the third step (see Sect. 4.3) of our method.

7 Conclusion

In this paper, we have proposed a privacy-preserving method for creating sanitized view of a data cubes. Our approach is based on data cube compression (by cell suppression), using a LP model that allows for the best cell deletion while maintaining maximum utility. Given a set of parameters, we also proposed a perturbation algorithm that is able to balance utility and privacy. We conducted extensive experimental tests to evaluate our approach, which was found to give better performances in terms utility and privacy.

In future work, we plan to explore other aspects of privacy to further develop our approach and compare it to well-known standards such as Differential Privacy.

References

1. Agrawal, R., Srikant, R., Thomas, D.: Privacy preserving OLAP. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 251–262 (2005)

2. Chatenoux, B., et al.: The Swiss data cube, analysis ready data archive using earth observations of Switzerland. Sci. Data **8**(1), 1–11 (2021)
3. Cuzzocrea, A., Saccà, D.: A theoretically-sound accuracy/privacy-constrained framework for computing privacy preserving data cubes in OLAP environments. In: Meersman, R., et al. (eds.) OTM 2012. LNCS, vol. 7566, pp. 527–548. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33615-7_6
4. Domingo-Ferrer, J., Sánchez, D., Blanco-Justicia, A.: The limits of differential privacy (and its misuse in data release and machine learning). Commun. ACM **64**(7), 33–35 (2021)
5. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**(3–4), 211–407 (2014)
6. Grubbs, P., Lacharité, M.-S., Minaud, B., Paterson, K.G.: Pump up the volume: practical database reconstruction from volume leakage on range queries. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 315–331 (2018)
7. Hylkema, M.: A survey of database inference attack prevention methods. Educational Technology Research (2009)
8. Kim, S., Lee, H., Chung, Y.D.: Privacy-preserving data cube for electronic medical records: an experimental evaluation. Int. J. Med. Inform. **97**, 33–42 (2017)
9. Nativi, S., Mazzetti, P., Craglia, M.: A view-based model of data-cube to support big earth data systems interoperability. Big Earth Data **1**(1–2), 75–99 (2017)
10. Sarathy, R., Muralidhar, K.: Evaluating Laplace noise addition to satisfy differential privacy for numeric data. Trans. Data Priv. **4**(1), 1–17 (2011)
11. Sung, S.Y., Liu, Y., Xiong, H., Ng, P.A.: Privacy preservation for data cubes. Knowl. Inf. Syst. **9**(1), 38–61 (2006). https://doi.org/10.1007/s10115-004-0193-2
12. Sweeney, L.: k-anonymity: a model for protecting privacy. Int. J. uncertainty Fuzziness Knowl.-Based Syst. **10**(05), 557–570 (2002)
13. Wang, L., Jajodia, S., Wijesekera, D.: Securing OLAP data cubes against privacy breaches. In: IEEE Symposium on Security and Privacy, Proceedings 2004, pp. 161–175. IEEE (2004)
14. Wang, L., Jajodia, S., Wijesekera, D.: Parity-based inference control for range queries. In: Wang, L., Jajodia, S., Wijesekera, D. (eds.) Preserving Privacy in On-Line Analytical Processing (OLAP). ADIS, vol. 29, pp. 91–117. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-46274-5_6
15. Wang, L., Jajodia, S., Wijesekera, D.: Preserving Privacy in On-Line Analytical Processing (OLAP), vol. 29. Springer, New York (2007). https://doi.org/10.1007/978-0-387-46274-5
16. Wang, L., Wijesekera, D., Jajodia, S.: Cardinality-based inference control in data cubes. J. Comput. Secur. **12**(5), 655–692 (2004)
17. Zhang, N., Zhao, W.: Privacy-preserving OLAP: an information-theoretic approach. IEEE Trans. Knowl. Data Eng. **23**(1), 122–138 (2010)