Cardinality-based inference control in data cubes

Lingyu Wang **, Duminda Wijesekera and Sushil Jajodia
Center for Secure Information Systems, George Mason University
MSN4A4, 4400 University Drive, Fairfax, VA 22030-4444, USA
E-mail: {lwang3,dwijesek,jajodia}@gmu.edu

This paper addresses the inference problem in on-line analytical processing (OLAP) systems. The inference problem occurs when the exact values of sensitive attributes can be determined through answers to OLAP queries. Most existing inference control methods are computationally expensive for OLAP systems, because they ignore the special structures of OLAP queries. By exploiting such structures, we derive cardinality-based sufficient conditions for safe OLAP data cubes. Specifically, data cubes are safe from inferences if their core cuboids are dense enough, in the sense that the number of known values is under a tight bound. We then apply the sufficient conditions on the basis of a three-tier inference control model. The model introduces an aggregation tier between data and queries. The aggregation tier represents a collection of safe data cubes that are pre-computed over a partition of the data using the proposed sufficient conditions. The aggregation tier is then used to provide users with inference-free queries. Our approach mitigates the performance penalty of inference control, because partitioning the data yields smaller input to inference control algorithms, pre-computing the aggregation tier reduces on-line delay, and using cardinality-based conditions guarantees linear-time complexity.

1. Introduction

Decision support systems such as On-line Analytical Processing (OLAP) are becoming increasingly important in industry. OLAP systems assist users in exploring trends and patterns in large amount of data by providing them with interactive results of statistical aggregations. Contrary to this initial objective, inappropriate disclosure of sensitive data stored in the underlying data warehouses results in the breach of individual’s privacy and jeopardizes the organization’s interest. It is well known that access control alone is insufficient in controlling information disclosure, because information not released directly may be inferred indirectly by manipulating legitimate queries about aggregated information, which is known as the inference control problem in databases. OLAP systems are especially vulnerable to such unwanted inferences, because of the aggregations used in OLAP queries. Providing inference-free answers to data cube style OLAP queries while not adversely impacting the response time of an OLAP system is the subject of this paper.

The inference problem has been investigated since 70’s with many inference control methods proposed, especially for statistical databases. However, most of those

---

*This work was partially supported by the National Science Foundation under grant CCR-0113515.
**Corresponding author. Tel.: +1 703 993-1629; Fax: +1 703 993-1638; E-mail: lwang3@gmu.edu.

0926-227X/04/$17.00 © 2004 – IOS Press and the authors. All rights reserved
methods become computationally infeasible if directly applied to OLAP systems. Most OLAP applications demand instant responses to interactive queries, although those queries usually aggregate a large amount of data [19,27]. Most existing inference control algorithms have run times proportional to the size of the queries or data set. Furthermore, those algorithms are enforced only after queries arrive, which makes it difficult to shift the computational complexity to off-line processing. The performance penalty renders them impractical for OLAP systems.

It has been pointed out that one way to make inference control practical is to restrict users to statistically meaningful queries [9]. Unlike in statistical databases where ad-hoc queries are quite common, queries having rich structures are usually more meaningful to OLAP users. In this paper we consider queries composed of the data cube operator introduced in [26]. The data cube operator generalizes many common OLAP operations such as group-by, cross-tab and sub-totals. As we shall show in this paper, efficient inference control is possible by exploiting special structures of the data cube operator.

Table 1 shows a data cube represented by four cross tabulations [26]. Each cross tabulation corresponds to a quarter of the year. The two dimensions are month and employee. Each internal cell of a cross tabulation contains the monthly commission of an employee. Assume that individual commissions are sensitive and should be hidden from users, and therefore have been replaced with “?”.

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Month/Employee</th>
<th>Alice</th>
<th>Bob</th>
<th>Jim</th>
<th>Mary</th>
<th>Sub total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>January</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>5500</td>
</tr>
<tr>
<td></td>
<td>Sub total</td>
<td>3000</td>
<td>3000</td>
<td>4500</td>
<td>6000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>April</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>6100</td>
</tr>
<tr>
<td></td>
<td>Sub total</td>
<td>4500</td>
<td>3300</td>
<td>4500</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sub total</td>
<td>3500</td>
<td>2200</td>
<td>2500</td>
<td>6000</td>
<td></td>
</tr>
<tr>
<td>* Bonus</td>
<td></td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>6000</td>
</tr>
<tr>
<td></td>
<td>Sub total</td>
<td>7000</td>
<td>4300</td>
<td>3000</td>
<td>7000</td>
<td></td>
</tr>
</tbody>
</table>
indicates that the employee is on leave and does not have a commission in the month. Each external cell of a cross tabulation contains either the subtotal commission of the four employees in a month, or the subtotal commission of an employee in a quarter.

Suppose that a malicious user Mallory wants to learn the hidden values represented by “?” in Table 1. Mallory can obtain knowledge about the table through two different channels: queries and the external knowledge (that is, the knowledge obtained through channels other than queries [17]). Firstly, she can ask legitimate queries about the subtotals in Table 1. Secondly, she knows the positions of empty cells in the table, because she works in the same company and knows which employee is on leave in which month. Now the inference problem occurs if any of the hidden values represented by “?” can be determined by Mallory. The following observations are relevant in this respect:

1. In the first and second quarter, no hidden value can be determined by Mallory, because infinitely many choices exist for any of them with all the subtotals satisfied. To illustrate, consider the first quarter. Suppose Mallory can determine a unique value $x_1$ for Alice’s commission in January. Then this value should not change with the choices of other values. Now let $S$ be a set of values satisfying the subtotals, in which Alice’s commission in February is $x_2$, and Bob’s commission in January and February are $y_1$ and $y_2$ respectively, as shown in the upper cross tabulation in Table 2. Now we can derive a different set of values $S'$ from $S$ by replacing $x_1$ with $x_1 - 100$, $x_2$ with $x_2 + 100$, $y_1$ with $y_1 + 100$ and $y_2$ with $y_2 - 100$, as shown in the lower tabulation in table 2. $S'$ also satisfies all the subtotals in quarter one, which implies that Alice’s commission in January cannot be determined by Mallory.

2. For the third quarter, Mary’s commission in September can be determined by Mallory as 2000, equal to the subtotal in September, because Mallory knows from external knowledge that Mary is the only employee who works and draws a commission in that month.

3. For the fourth quarter, no hidden value can be determined in the similar way as in the third quarter, because all the subtotals in the fourth quarter are calculated

<table>
<thead>
<tr>
<th>Quarter</th>
<th>Month / Employee</th>
<th>Alice</th>
<th>Bob</th>
<th>Jim</th>
<th>Mary</th>
<th>Sub total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>January</td>
<td>$x_1$</td>
<td>$y_1$</td>
<td>?</td>
<td>?</td>
<td>5500</td>
</tr>
<tr>
<td></td>
<td>February</td>
<td>$x_2$</td>
<td>$y_2$</td>
<td>?</td>
<td>?</td>
<td>5500</td>
</tr>
<tr>
<td></td>
<td>Sub total</td>
<td>3000</td>
<td>3000</td>
<td>4500</td>
<td>6000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>January</td>
<td>$x_1 - 100$</td>
<td>$y_1 + 100$</td>
<td>?</td>
<td>?</td>
<td>5500</td>
</tr>
<tr>
<td></td>
<td>February</td>
<td>$x_2 + 100$</td>
<td>$y_2 - 100$</td>
<td>?</td>
<td>?</td>
<td>5500</td>
</tr>
<tr>
<td></td>
<td>Sub total</td>
<td>3000</td>
<td>3000</td>
<td>4500</td>
<td>6000</td>
<td></td>
</tr>
</tbody>
</table>
from at least two hidden values. However, the following inference is possible. As shown in Table 3, let $x_1$ be Alice’s commission in October; let $y_1$ and $y_2$ be Bob’s commission in October and November respectively; and let $z_1$ and $z_2$ be Jim’s commission in October and November respectively. Mallory asks four legitimate queries:

(a) $x_1 + y_1 + z_1 = 7100$ (The subtotal commission in October)
(b) $y_2 + z_2 = 4100$ (The subtotal commission in November)
(c) $y_1 + y_2 = 4300$ (Bob’s total commission in this quarter)
(d) $z_1 + z_2 = 3000$ (Jim’s total commission in this quarter)

By adding both sides of the first two equations (a) and (b), and then subtracting from the result the last two equations (c) and (d), Mallory gets $x_1 = 3900$, which is Alice’s commission in October.

To generalize the above example, unknown variables and their aggregations are used to represent commissions and subtotals of commissions, respectively. Empty cells are used to represent the values that users already learn from external knowledge. A data cube is compromised if the value of any unknown variables can be uniquely determined from the aggregations and the empty cells. In order to efficiently determine if a given data cube is compromised, we derive cardinality-based sufficient conditions for safe data cubes. Specifically, we show that any data cube is safe from compromises if the number of empty cells is below a given bound. The bound is tight because any data cube having more empty cells than the bound is subject to compromises. We apply the sufficient conditions on the basis of a three-tier inference control model. Besides data and queries, the model introduces a new tier, which represents a collection of safe data cubes. Using the proposed sufficient conditions, the safe data cubes are first computed over a partition of the data, and then used to provide users with inference-free queries. The overhead of inference control in terms of response time is mitigated by such an approach, because partitioning the data yields smaller input to inference control algorithms, pre-computing the aggregation tier reduces on-line delay, and using cardinality-based sufficient conditions for computation guarantees linear-time complexity. In this paper we elaborate and justify the preliminary results given in [41], and address various implementation issues and improvements to the algorithm.
The rest of the paper is organized as follows. Section 2 reviews existing inference control methods proposed in statistical databases. Section 3 formalizes sum-only data cubes and the compromises. Section 4 proves cardinality-based sufficient conditions for safe data cubes. Section 5 proposes a three-tier inference control model. Section 6 integrates the sufficient conditions of safe data cubes into an inference control algorithm on the basis of the three-tier model. Section 7 concludes the paper.

2. Related work

Inference control has been extensively studied in statistical databases as surveyed in [1, 17, 18]. Inference control methods proposed in statistical databases are usually classified into two main categories: restriction based techniques and perturbation based techniques. Restriction based techniques [25] include restricting the size of a query set (that is, the values aggregated by a single query), overlap control [21] in query sets, auditing all queries in order to determine when inferences are possible [6, 10, 13, 29, 33], suppressing sensitive data in released statistical tables [14, 22], partitioning data into blocks and treating each block as a single value in answering queries [11, 12, 34]. Perturbation based technique includes adding noise to source data [39], outputs [5], database structure [36], or size of query sets (that is, answering queries using sampled data) [16]. Some variations of the inference problem have been studied lately, such as the inference caused by arithmetic constraints [7, 8], the inference of approximate values instead of exact values [32] and the inference of small intervals enclosing exact values [30]. Finally, techniques similar to suppression and partitioning are being developed to protect the sensitive information in released census data [14, 22, 34, 37, 42, 43].

Our work shares with [21] the similar motivation of controlling inferences with cardinalities of data and queries. Dobkin et al. give an lower bound on the number of ad-hoc sum queries that compromise sensitive values [21]. Under the assumptions that each query sums exactly \( k \) sensitive values and no two queries sum more than \( r \) values in common, Dobkin et al. prove that at least \( 1 + (k - 1)/r \) sum queries are required to compromise any sensitive value. Although this result certainly applies to
OLAP queries, it does not take into account the special structures of those queries. The restricted form of queries imply better results (that is, tighter bounds). In this paper we pursue such better results by exploiting the special structures of OLAP queries.

Recently a variation of the inference control problem, namely, **privacy preserving data mining**, has drawn considerable attention as seen in [2,3,20,23,24,35,40]. Similar to perturbation-based techniques used in statistical databases, random perturbation is used to hide sensitive values. The statistical distribution model of those values is preserved during perturbations, such that data mining results such as classifications or association rules can be obtained from the perturbed data. In contrast to data mining, the responsibility of OLAP systems is to provide users with the means (that is, precise answers to OLAP queries), instead of the results (the discovered trends or patterns). Preserving the distribution model of sensitive values is insufficient for OLAP applications, because the precision of answers to OLAP queries is not guaranteed. To our understanding, introducing enough noises to protect sensitive data while avoiding bias or inconsistence in the answers to OLAP queries is still an open problem. Our work is not based on perturbation, but based on the restriction of unsafe queries. Although all queries are not answered as required by inference control, the answers are always precise.

In [41] we present some preliminary results including the cardinality-based sufficient conditions for safe data cubes. In this paper we expand those results with more detailed justification and implementation consideration. We relax the assumptions about missing tuples [41], such that our work can deal with general external knowledge of sensitive values. We clarify the sufficient conditions by providing both underlying intuitions and formal justifications. We also address implementation issues such as incrementally updating the results of inference control algorithms once data are inserted or deleted.

3. Preliminaries

This section defines the basic notations. First in Section 3.1 we define the components of **data cubes**. Next in Section 3.2 we define **aggregation matrix** and **compromisability**. Finally in Section 3.3 we explain the choices made in our design.

3.1. Data cube

A data cube is composed of a **core cuboid** and a collection of **aggregation cuboids**. Corresponding to the discussion in Section 1, the core cuboid essentially captures the notion of sensitive values and empty cells, while the aggregation cuboids correspond to the subtotals.

Definition 1 formalizes the concepts related to the core cuboid. We use closed integer intervals for **dimensions**. The Cartesian product of \( k \) dimensions form the **full**
core cuboid. We call each vector in the full core cuboid a tuple. Hence the full core cuboid is simply the collection of all possible tuples that can be formed by the $k$ dimensions. A core cuboid needs not include all possible tuples, but has to satisfy the property that any integer from any dimension must appear in at least one tuple in the core cuboid. This property ensures that any given core cuboid corresponds to a unique full core cuboid (as well as the $k$ dimensions). The tuples not appearing in a core cuboid are said to be missing. By fixing a value for one of the $k$ dimensions, we can select tuples from a core cuboid to form a slice on that dimension. A slice is full if no tuple is missing from it. Clearly a slice can be full even if the core cuboid cannot.

**Definition 1 (Core Cuboids).**

1. Given $k$ ($k > 1$) integers $d_1, d_2, \ldots, d_k$ satisfying $d_i > 1$ for all $1 \leq i \leq k$, the $i$th dimension, denoted by $D_i$, is the closed integer interval $[1, d_i]$.

2. The $k$-dimensional full core cuboid, denoted as $C_{\text{full}}$, is the Cartesian product $\prod_{i=1}^{k} D_i$. Each vector $t \in C_{\text{full}}$ is referred to as a tuple.

3. A $k$-dimensional core cuboid is any $C_{\text{core}} \subseteq C_{\text{full}}$ satisfying that $\forall i \in [1, k] \ \forall x \in D_i, \exists t \in C_{\text{core}} \ t[i] = x$. Notice that we use notation $t[i]$ for the $i$th element of vector $t$ from now on.

4. $t$ is missing if $t \in C_{\text{full}} \setminus C_{\text{core}}$.

5. The $j$th ($j \in D_j$) slice of $C_{\text{core}}$ on the $i$th dimension, denoted by $P_i(C_{\text{core}}, j)$, is the set $\{ t : t \in C_{\text{core}}, t[i] = j \}$. $P_i(C_{\text{core}}, j)$ is full if $P_i(C_{\text{core}}, j) = P_i(C_{\text{full}}, j)$.

Table 5 gives an example to the concepts in Definition 1. The example describes a two dimensional core cuboid, with both dimensions as $[1, 4]$ (we use normal font for the first dimension and italic for the second for clarity purpose). The upper left tabulation shows the full core cuboid $[1, 4] \times [1, 4]$. The upper right tabulation shows a core cuboid with nine tuples. As required by Definition 1, any integer from any dimension appears in at least one of the nine tuples. Without this restriction, one could have argued that the first dimension is $[1, 5]$ instead of $[1, 4]$, and hence the full core cuboid contains $5 \times 4 = 20$ tuples instead of 16. This situation must be avoided in order to use the cardinalities. The left lower cross tabulation shows the seven missing tuples. The right lower cross tabulation shows the first slice on the first dimension. The core cuboid in Table 5 is analogous to the fourth cross tabulation in Table 1 in the following sense. The employee names and months in Table 1 are abstracted as integers from one to four. The hidden values represented in "?" in Table 1 become integer vectors, and the empty cells correspond to missing tuples. The commissions in October are now called the first slice on the first dimension.

Definition 2 formalizes the concepts related to aggregation cuboids. The *-value is a special value unequal to any integers. *-values appearing in a vector are called *-elements. Adding the *-value into any dimension yields an augmented dimension.
The Cartesian product of the \( k \) augmented dimensions includes all the vectors whose elements are either integers (from the dimensions), or *-elements. A vector with no *-element is simply a tuple, and a vector with \( j \) many *-elements are called a \( j-* \) aggregation vector. The collection of all the \( j-* \) aggregation vectors having *-elements at the same positions is called a \( j-* \) aggregation cuboid. Any aggregation vector can be used to match tuples in the core cuboid, with a *-element matching any integers and an integer matching only itself. The matched tuples form the aggregation set of the aggregation vector. A data cube is a pair of the core cuboid and the collection of all aggregation cuboids.

**Definition 2 (Aggregation Cuboids and Data Cubes).**

1. A \(*-value\), denoted as \(*\), is a special value unequal to any positive integer. A *-value appearing in a vector is called a \(*-element\).
2. Given the \( i\)th dimension \( D_i \), the \( i\)th augmented dimension, denoted as \( D_i^* \), is \([1, d_i] \cup \{\ast\}\).
3. A \( j-* \) aggregation cuboid is the maximal subset of the Cartesian product \( \Pi_{i=1}^k D_i^* \) satisfying,
   
   (a) \( \forall i \in [1, k] \forall t, t' \in C_{aggr}, t[i] = \ast \iff t'[i] = \ast \); and
   
   (b) \( \forall t \in C_{aggr}, |\{i : t[i] = \ast\}| = j \).

   Each vector \( t \in C_{aggr} \) is called a \( j-* \) aggregation vector.
4. Given \( C_{core} \), the aggregation set of any aggregation vector \( t_{aggr} \), denoted as \( Q_{set}(t_{aggr}) \), is the set of tuples: \( \{t : t \in C_{core}, \forall i \in [1, k], t_{aggr}[i] \neq \ast \Rightarrow t[i] = t_{aggr}[i]\} \). The aggregation set of a set of aggregation vectors \( C \), denoted as \( Q_{set}(C) \), is the set of tuples: \( \cup_{t_{aggr} \in C} Q_{set}(t) \).
5. A \( k\)-dimensional data cube is a pair \(<C_{core}, S_{all}>\), where \( C_{core} \) is any core cuboid with dimensions \( D_1, D_2, \ldots, D_k \) and \( S_{all} \) is the set of all aggregation cuboids with dimensions \( D_1, D_2, \ldots, D_k \).

Table 4 gives an example to the concepts in Definition 2. The two augmented dimensions are both \([1, 2, 3, 4, \ast]\). As shown in the left cross tabulation in Table 4, the Cartesian product of the two augmented dimensions yields 25 vectors. There are two \( 1-* \) aggregation cuboids: \( \{(1, \ast), (2, \ast), (3, \ast), (4, \ast)\} \) and \( \{\ast, 1\}, (\ast, 2\}, (\ast, 3\}, (\ast, 4\} \), and one \( 2-* \) aggregation cuboids \( \{\ast, \ast\} \). The right cross tabulation in Table 4 shows a 2-dimensional data cube. Notice that the two augmented dimensions are included for the purpose of clarity, and they are not a part of the data cube. As an example of aggregation set, the tuples composing the aggregation set of \( (1, \ast) \) are underlined. The \( 1-* \) aggregation vectors in Table 4 abstract the subtotals in the fourth cross tabulation in Table 1, and the \( 2-* \) aggregation vector corresponds to the total commission in the fourth quarter.
Table 4
Illustration of aggregation cuboids and data cube

| Cartesian Product \{1, 2, 3, 4, *\} \times \{1, 2, 3, 4, *\} |
|---|---|---|---|---|
| 1 | (1,1) | (1,2) | (1,3) | (1,4) | (1,*) |
| 2 | (2,1) | (2,2) | (2,3) | (2,4) | (2,*) |
| 3 | (3,1) | (3,2) | (3,3) | (3,4) | (3,*) |
| 4 | (4,1) | (4,2) | (4,3) | (4,4) | (4,*) |
| * | (*,1) | (*,2) | (*,3) | (*,4) | (*,*) |

A data cube

| 1 | (1,1) | (1,2) | (1,3) | (1,4) | (1,*) |
| 2 | (2,1) | (2,2) | (2,3) | (2,4) | (2,*) |
| 3 | (3,1) | (3,2) | (3,3) | (3,4) | (3,*) |
| 4 | (4,1) | (4,2) | (4,3) | (4,4) | (4,*) |
| * | (*,1) | (*,2) | (*,3) | (*,4) | (*,*) |

3.2. Compromisability

We first define aggregation matrix, and then formalize compromisability. In order to characterize a data cube, it suffices to know which tuple is a member of the aggregation set of which aggregation vector. Aggregation matrix captures this membership relation in a concise manner.

In order to fix notation, we use the following conventions in our further discussions of sets of vectors. Whenever applicable, we assume the members of a set are sorted according to the orders stated below:

1. Tuples in a core cuboid (or its subset) and aggregation vectors in an aggregation cuboid (or its subset) are in dictionary order (by saying so, we are treating vectors as strings with the leftmost element the most significant). For example, the core cuboid \(C_{core}\) in Table 5 is sorted as \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 1), (3, 4), (4, 1), (4, 4)\}.

2. Aggregation cuboids in \(S_{all}\) or its subsets are sorted first in ascending order according to the number of *-elements in their aggregation vectors, and then in descending order on the index of the *-elements. For example, \(S_{all}\) shown in Table 4 is sorted as \{\{(1, *), (2, *), (3, *), (4, *)\}, \{(*, 1), (*, 2), (*, 3), (*, 4)\}, \{(*, *)\}\}.

3. We use notation \(C[i]\) for the \(i\)th member of the sorted set \(C\).

Definition 3 formalizes aggregation matrix. Suppose the full core cuboid includes \(n\) tuples, and we are given \(m\) aggregation vectors, then the aggregation matrix of those aggregation vectors is an \(m\) by \(n\) matrix \(M\). Each element of the aggregation matrix \(M\) is either one or zero, and \(M[i, j] = 1\) if and only if the \(j\)th tuple in
Table 5
Example of a core cuboid with \( k = 2 \) and \( d_1 = d_2 = 4 \)

<table>
<thead>
<tr>
<th>The Full Core Cuboid ( C_{\text{full}} )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   ( (1,1) )</td>
<td>(1,2)</td>
<td>(1,3)</td>
<td>(1,4)</td>
<td></td>
</tr>
<tr>
<td>2   ( (2,1) )</td>
<td>(2,2)</td>
<td>(2,3)</td>
<td>(2,4)</td>
<td></td>
</tr>
<tr>
<td>3   ( (3,1) )</td>
<td>(3,2)</td>
<td>(3,3)</td>
<td>(3,4)</td>
<td></td>
</tr>
<tr>
<td>4   ( (4,1) )</td>
<td>(4,2)</td>
<td>(4,3)</td>
<td>(4,4)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A Core Cuboid ( C_{\text{core}} )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1   ( (1,1) )</td>
<td>(1,2)</td>
<td>(1,3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2   ( (2,2) )</td>
<td>( (2,3) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3   ( (3,1) )</td>
<td>( (3,4) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4   ( (4,4) )</td>
<td>( (4,4) )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Missing Tuples ( C_{\text{full}} \setminus C_{\text{core}} )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( (1,4) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( (2,1) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( (3,2) )</td>
<td>( (3,3) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>( (4,2) )</td>
<td>( (4,3) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1st Slice on 1st Dimension ( P_1(C_{\text{core}}, 1) )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( (1,1) )</td>
<td>(1,2)</td>
<td>(1,3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the full core cuboid is included in the aggregation set of the \( i \)th aggregation vector. Intuitively, a column of \( M \) stands for a tuple, a row for an aggregation vector, and an element of 1 for a matching between them. An element of 0 could mean two things: either the tuple is missing or it is in the core cuboid but does not match the aggregation vector. The aggregation matrix is unique if the above convention of ordering is followed.

**Definition 3 (Aggregation Matrix).**

1. In a given data cube \( < C_{\text{core}}, S_{\text{full}} > \), suppose \( |C_{\text{full}}| = n \) and let \( C_{\text{aggr}} \) be any set of \( m \) aggregation vectors. The **aggregation matrix** of \( C_{\text{aggr}} \) is the \( (m \times n) \) matrix \( M_{C_{\text{core}}, C_{\text{aggr}}} \):
$M_{C_{\text{core}}.C_{\text{agg}}}[i,j] = \begin{cases} 
1, & \text{if } C_{\text{full}}[j] \in Q \text{ set}(C_{\text{agg}}[i]); \\
0, & \text{otherwise.}
\end{cases}$

2. Given a set of sets of aggregation vectors $S$ (for example, $S_{\text{all}}$), $M_{C_{\text{core}}.S}$ is the row block matrix with the $i$th row block as the aggregation matrix of the $i$th set in $S$. Specially, we use $S_1$ for the set of all 1-* aggregation cuboids and $M_1$ for its aggregation matrix, referred to as the 1-* aggregation matrix.

Table 6 illustrates the concept of aggregation matrix. The cross tabulation shows the same data cube as in Table 4, with the tuples and aggregation vectors indexed with subscripts according to our order convention. For clarity purpose, normal font are used for the indexes of tuples while italic font for those of aggregation vectors. The 1-* aggregation matrix $M_1$ is shown in the lower part of Table 6. The rows and columns of the matrix are both indexed accordingly. As an example, the first row of $M_1$ contains three 1 elements, because the first aggregation vector $(1, *)$ has three tuples $(1, 1), (1, 2)$ and $(1, 3)$ in its aggregation set. The fourth column of $M_1$ is a zero column because the fourth tuple $(1, 4)$ in the full core cuboid is missing from the core cuboid.

Before we formally define compromisability, we first give the underlying intuitions. With the rows of an aggregation matrix $M$ corresponding to aggregation vectors, the elementary row operations on $M$ captures possible inferences users can
make with the aggregation vectors. For example, in the discussion of Table 1 in Section 1, the addition of two subtotals can be represented by adding two corresponding rows in the aggregation matrix. The compromise of a tuple can then be captured by a sequence of elementary row operations that leads to a unit row vector (that is, a row vector having a single 1 element and all other elements as 0). This unit row vector represents an aggregation of a single tuple, which is compromised. From linear algebra [28], such a sequence of elementary row operations exists for a matrix if and only if the reduced row echelon form (RREF) of the matrix has at least one unit row vector.

Definition 4 formalizes the compromisability based on above intuitions. The compromisability is decidable for any given set of aggregation vectors, because the RREF of any matrix is unique and can be obtained through a finite number of elementary row operations. Trivial compromises occur when any of the given aggregation vectors already compromises the core cuboid (that is, it aggregates a single tuple). In the absence of trivial compromises, one has to manipulate more than one aggregation vector to compromise any tuple. This is called non-trivial compromises.

Definition 4 (Compromisability).

1. In a given data cube \(< C_{\text{core}}, S_{\text{all}} \>\), let \( C_{\text{aggr}} \) be any set of aggregation vectors, \( C_{\text{aggr}} \) compromises \( C_{\text{core}} \) if there exists at least one unit row vector in the reduced row echelon form (RREF) of \( M_{C_{\text{core}}, C_{\text{aggr}}} \).
2. Suppose \( C_{\text{aggr}} \) compromises \( C_{\text{core}} \). We say \( C_{\text{aggr}} \) trivially compromises \( C_{\text{core}} \) if there exists at least one unit row vector in \( M_{C_{\text{core}}, C_{\text{aggr}}} \), and \( C_{\text{aggr}} \) non-trivially compromises \( C_{\text{core}} \), otherwise. We say that the \( i \)th tuple is compromised, if the RREF of \( M_{C_{\text{core}}, C_{\text{aggr}}} \) contains a unit row vector whose \( i \)th element is 1.

Table 7 gives an example of compromisability. The matrix in Table 7 is the RREF of the aggregation matrix \( M_1 \) in Table 6. The first row of \( M_1 \) is a unit row vector.

<table>
<thead>
<tr>
<th>A Data Cube With Tuples and Aggregation Vectors Indexed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
Hence in Table 6, the set of 1-* aggregation cuboids $S_1$ compromises the core cuboid $C_{\text{core}}$. Because $M_1$ contains one unit row vector $e_1$, the first tuple in $C_{\text{core}}$ is compromised. Moreover, this is a non-trivial compromise because the original aggregation matrix $M_1$ does not contain any unit row vector.

3.3. Formalization rationale

It is a common approach in the literature to model dimensions using integer intervals. The actual values of different data types in the domain of a dimension is related to a set of integers by a one-to-one mapping. For example, dimensions month and employee in Table 3 are mapped to integer intervals $[1, 4]$ in Table 5. Such an abstraction ignores the specific values in each dimension and focuses on the structure of the data cube. Although dimensions may have continuous domains and infinitely many values, such as commissions, any given instance of data cube must contain only finite number of values. As stated in Definition 1, we map a value to an integer only if it appears in one or more tuples in the core cuboid. Hence it is sufficient to use an arbitrarily large but fixed integer interval for each dimension, as in Definition 1. Notice that some inference problems depend on the specific values in each tuple, such as those discussed in [30,31,33]. We do not address those problems in this paper.

The core cuboid, aggregation cuboids, and data cube in Definition 1 are similar to those in [26]. For example, the real-world data cube in Table 3 is modeled in Table 5. However, unlike in [26], we model data cubes using sets of vectors rather than using a relational operator. In doing so we are able to conveniently refer to any specific tuple or aggregation without complex relational queries. This choice simplifies our notation. We define the core cuboid and the aggregation cuboid separately, while they are not explicitly distinguished in [26]. The reason for our choice is that only tuples in core cuboid are assumed to contain sensitive values, but aggregations are not. This may not be valid for some special applications, where some users are prohibited from accessing certain aggregated values as well. Our ongoing work address this issue.

The missing tuples formalized in Definition 1 reflect the external knowledge of users (that is, the knowledge obtained through channels other than queries). One example of external knowledge is the unpopulated cells (that is, combinations of dimension values) in a sparse data cube. Users usually know which cells of the data cube are populated and which are not. This is so if the dimension values of each tuple is made public since they are not sensitive. One may argue that those values should be kept secret, because then inferences become impossible. However, even if all the dimension values are hidden, users may still infer the positions of unpopulated cells through queries containing COUNT. Preventing inferences through COUNT queries has been shown as intractable [29]. Hence we make the safer assumption that the positions of unpopulated cells are public.

Our definition of missing tuples captures a broader concept of external knowledge than unpopulated cells in a data cube. In practice users may learn values of sensitive
attributes through many channels. We use missing tuples to characterize such known values, regardless of the specific channels in which they are learned. From the viewpoint of both malicious users and inference control, values become irrelevant once they are learned through external knowledge, and hence are removed from the core cuboid. The specific values being learned are also irrelevant. For example, Table 8 shows two variations of the data cube shown in Table 3. The upper cross tabulation assumes that users know employee Mary to have a zero (but valid) commission for October. The lower cross tabulation assumes that users know Mary to have a commission of 1000. For those two varied data cubes, no change is necessary for the model given in Table 5.

Our definition of aggregation matrix and compromisability is similar to those used by Chin [13]. In [13], a set of sum queries over sensitive values is modeled as a linear system of equations. It then follows that determining compromisability of the sensitive values is equivalent to determining the existence of unit row vectors in the RREF of the coefficient matrix of the linear system. In our study, we directly define the compromisability based on the RREF of the aggregation matrix without referring to this well known equivalence. We distinguish between trivial and non-trivial compromises to facilitate our study, because they exhibit different cardinality-based characterizations as we shall show in the following sections. In the literature trivial and non-trivial compromises are referred to as small query set attack (or single query attack) [15], and linear system attack (or multiple query attack) [21], respectively.

4. Cardinality-based sufficient conditions for inference-free data cubes

In this section we prove cardinality-based sufficient conditions for safe data cubes. Those conditions relate the cardinality of core cuboids to the compromisability of
data cubes. We discuss trivial compromises in Section 4.1 and non-trivial compromises in Section 4.2. As stated in Section 3, the difference between the two cases is whether or not compromises can be achieved with a single aggregation.

4.1. Trivial compromisability

We have two results for trivial compromises as stated in Theorem 1. The first says that full core cuboids cannot be trivially compromised. This holds because in a full core cuboid all aggregation sets are larger than one. First consider the 1-* aggregation vectors. The aggregation set of any 1-* aggregation vector is equal to the size of the dimension having *-value, which is larger than one. Moreover, the aggregation set of any aggregation vector with more than one *-elements has a cardinality no less than some 1-* aggregation vectors. Hence any aggregation set contains more than one tuple. For example, in Table 4, the aggregation set of all the 1-* aggregation vectors contain at least two tuples, and the only 2-* aggregation vector contains all the nine tuples in the core cuboid. The second claim of Theorem 1 states that any core cuboid containing fewer tuples than the given upper bound is always trivially compromised. Intuitively, this is so because not enough tuples exist in the core cuboid in order for all the aggregation sets to contain more than one tuple. Notice if the core cuboid contains no tuple at all, then no aggregation set contains one tuple. However, this extreme case does not conform to Definition 1, because we require all dimension values to be present in at least one tuple.

**Theorem 1.** In a given k dimensional data cube <C_{core}, S_{all}> with dimensions \( D_1, D_2, \ldots, D_k \), we have:

1. \( C_{full} \) cannot be trivially compromised by any aggregation cuboid \( C \in S_{all} \).
2. \( C_{core} \) is trivially compromised by \( S_i \) if \( |C_{core}| < 2^{k-1} \cdot \max(d_1, d_2, \ldots, d_k) \).

**Proof.**

1. By Definition 4 we need to show that for any \( t \in C \), we have \(|Q\text{ set}(t)| > 1\). Without loss of generality, let \( t \) be the j-* aggregation vector \( (*, \ldots, *, x_{j+1}, x_{j+2}, \ldots, x_k) \). By Definition 2, we have \( Q\text{ set}(t) = \{ t' : t' \in C_{full}, t'[j+1] = x_{j+1}, t'[j+2] = x_{j+2}, \ldots, t'[k] = x_k \} \). Because \( C_{full} = \Pi_{i=1}^k [1, d_i] \) we have \(|Q\text{ set}(t)| = \prod_{i=1}^k d_i \). Because \( d_i > 1 \) for all \( 1 \leq i \leq k \), we have \(|Q\text{ set}(t)| > 1\).

2. Suppose \( C_{core} \) is not trivially compromised. We show that \(|C_{core}| \geq 2^{k-1} \cdot \max(d_1, d_2, \ldots, d_k) \). Without loss of generality, assume \( d_k \geq D_i \) for all \( 1 \leq i \leq k \). By Definition 1, there are totally \( d_k \) slices of \( C_{core} \) on the \( k \)th dimension. Without loss of generality it suffices to show that \(|P_k(C_{core}, 1)| \geq 2^{k-1}\). We do so by mathematical induction on \( i \leq k \) as given below.

**The Inductive Hypothesis:** For \( i = 1, 2, \ldots, k \), there exists \( C_i \subseteq P_k(C_{core}, 1) \) satisfying \(|C_i| = 2^{i-1}\), and for any \( t_1, t_2 \in C_i \) we have \( t_1[j] = t_2[j] \) for all \( j \geq i \).
The Base Case: By Definition 1, there exists \( t \in C_{\text{core}} \) satisfying \( t[k] = 1 \). Let \( C_1 \) be \( \{t\} \). We have \( C_1 \subseteq P_k(C_{\text{core}}, 1) \) and \( |C_1| = 1 \), validating the base case of our inductive hypothesis.

The Inductive Case: Suppose for all \( 1 \leq i < k \) there exists \( C_i \subseteq P_k(C_{\text{core}}, 1) \) satisfying \( |C_i| = 2^{i-1} \), and for any \( t_1, t_2 \in C_i \), \( t_1[j] = t_2[j] \) for all \( j \geq i \). We show that there exists \( C_{i+1} \subseteq P_k(C_{\text{core}}, 1) \) such that \( |C_{i+1}| = 2^i \), and for any \( t_1, t_2 \in C_{i+1} \), \( t_1[j] = t_2[j] \) for all \( j \geq i + 1 \).

For any \( t_j \in C_i \), let \( t_j' = t_j \) and \( t_j'' = t_j \) for all \( j \neq i \). Because \( t_j \in Q \), we have \( |Q \cup \{t_j\}| = 1 \). Since \( C_{\text{core}} \) is not trivially compromised by \( S_1 \), according to Definition 4, we have \( |Q \cup \{t_j\}| > 1 \). Hence, there exists \( t_j' \in Q \cup \{t_j\} \subseteq C_{\text{core}} \) such that \( t_j'[i] \neq t_j[i] \) and \( t_j''[j] = t_j[j] \) for all \( j \neq i \); which implies \( t_j' \notin C_i \).

Similarly for any \( t_j \in C_i \) satisfying \( t_1 \neq t_2 \), there exists \( t_j'' \in C_{\text{core}} \) such that \( t_j''[i] \neq t_j[i] \) and \( t_j''[j] = t_j[j] \) for all \( j \neq i \). Now we show that \( t_j'' \neq t_j' \).

Because \( t_1[j] = t_2[j] \) for all \( j \geq i \), there must be \( l < i \) such that \( t_1[l] \neq t_2[l] \). Because \( t_j''[j] = t_j[j] \) and \( t_j''[j] = t_j[j] \) for all \( j < i \), we have that \( t_j''[l] \neq t_j'[l] \). That is, \( t_j'' \neq t_j' \).

Hence there exists \( C_i' \subseteq C_{\text{core}} \) satisfying \( |C_i'| = |C_i| \), and for any \( t \in C_i \), there exists one and only one \( t' \in C_i' \) such that \( t[i] \neq t'[i] \) and \( t[j] = t'[j] \) for all \( j \neq i \). Let \( C_{i+1} \) be \( C_i \cup C_i' \). Then \( |C_{i+1}| = 2^i \).

This proves the inductive case of our induction, from which the claim \( |P_k(C_{\text{core}}, 1)| \geq 2^{k-1} \) follows.

For data cubes with cardinalities between the two limits stated in Theorem 1, the trivial compromisability cannot be determined solely by the cardinality of core cuboids. Two data cubes whose core cuboids have the same cardinality but different missing tuples can have different trivial compromisability. For example, the core cuboid \( C_{\text{core}} \) in Table 4 is not trivially compromised. Without changing the cardinality of \( C_{\text{core}} \), we delete the tuple \( (2, 2) \) and add a new tuple \( (1, 4) \) to obtain a new core cuboid \( C'_{\text{core}} \). \( C'_{\text{core}} \) is trivially compromised although \( |C'_{\text{core}}| = |C_{\text{core}}| \), because in \( C'_{\text{core}} \) the aggregation sets of \( (2, +) \) and \( (+, 2) \) contain exactly one tuple.

The trivial compromisability of data cubes can also be determined by computing the cardinalities of all the rows in the aggregation matrix. With an \( m \times n \) aggregation matrix this can be done in \( O(mn) \). For example, for the aggregation matrix \( M_1 \) given in Table 6, counting the 1-elements in each row shows that \( C_{\text{core}} \) is not trivially compromised by \( S_1 \). The complexity can be further reduced considering that \( M_{i}[i, j] = 1 \) only if \( M_{C_{\text{core}}, S_i}[i, j] = 1 \). For example, to calculate the aggregation set \( Q \cup \{t_j\} \) for the aggregation vector \( t = (1, +) \), we only need to consider the four elements \( M_{i}[1, 1], M_{i}[1, 2], M_{i}[1, 3] \) and \( M_{i}[1, 4] \), as we know that \( M_{i}[1, j] = 0 \) for all \( j > 4 \).
4.2. Non-trivial compromisability

Our results for non-trivial compromises require the observations stated in Lemma 1. Intuitively, the first claim of Lemma 1 holds, because aggregation vectors in any single aggregation cuboid always have disjoint aggregation sets (that is, any tuple is aggregated by at most one aggregation vector), and hence they do not help each other in non-trivial compromises. The second claim of Lemma 1 holds because aggregation vectors having more than one *-value can be derived from some 1-* aggregation vectors. For example, in Table 4 the aggregation set of the 2-* aggregation vector \((*,*)\) is equal to the aggregation set of one 1-* aggregation cuboid. Because of the second claim, it is sufficient to consider \(S_1\) instead of \(S_{all}\) to determine the compromisability of data cubes. The last condition in Lemma 1 says that it is impossible to determine the non-trivial compromisability of a data cube by merely observing its \(k\) dimensions. This implies that any large enough data cube is vulnerable to non-trivial compromises. Here a data cube is large enough if \(d_i \geq 4\) for all \(1 \leq i \leq k\). For example, when \(k = 2\) and \(d_1 = d_2 = 2\), no data cube will be non-trivially compromisable.

Lemma 1. 1. In any given data cube \(<C_{core}, S_{all}>\), \(C_{core}\) can not be non-trivially compromised by any single cuboid \(C \in S_{all}\).
2. In any given data cube \(<C_{core}, S_{all}>\), if \(C_{core}\) cannot be compromised by \(S_1\), then it cannot be compromised by \(S_{all}\).
3. For any integers \(k\) and \(d_1, d_2, \ldots, d_k\) satisfying \(d_i \geq 4\) for \(1 \leq i \leq k\), there exists a \(k\)-dimensional data cube \(<C_{core}, S_{all}>\) with the \(k\) dimensions \(D_1, D_2, \ldots, D_k\), such that \(C_{core}\) is non-trivially compromised by \(S_1\).

Proof.

1. Let \(C \in S_{all}\) be any aggregation cuboid. For any \(t \in C_{core}\) there exists one and only one \(t_{aggr} \in C\) such that \(t \in Q_{set}(t_{aggr})\). Hence, in the aggregation matrix \(M\) each non-zero column is a unit column vector, implying that \(M\) could be transformed into its RREF by permuting its columns. Furthermore, each row of \(M\) must contain at least two 1’s because no trivial compromise is assumed. Hence no unit row vector is in the RREF of \(M\). Thus, \(C_{core}\) cannot be non-trivially compromised by \(C\).

2. Without loss of generality, let \(t_{aggr}\) be any \(j\)-* \((j > 1)\) aggregation vector satisfying that \(t_{aggr}[i] = *\) for any \(1 \leq i \leq j\). Let \(C\) be the set of \(1\)-* aggregation vectors defined as: \(\{t: t[1] = *, t[i] \neq * \forall i \in [2, j], t[i] = t_{aggr}[i] \forall i \in [j + 1, k]\}\). We have that \(Q_{set}(t_{aggr}) = Q_{set}(C)\). Hence in the aggregation matrix \(M_{C_{core}, S_{all}}\), the row corresponding to \(t_{aggr}\) can be represented as the linear combination of the rows corresponding to \(C\). The rest of the proof follows from linear algebra.
3. First we justify the case \( k = 2 \), then we extend the result to \( k > 2 \).

For the proof of \( k = 2 \), without loss of generality, we use mathematical induction on \( d_1 \), for an arbitrary, but fixed value of \( d_2 \geq 4 \).

**The Inductive Hypothesis:** For any \( d_1, d_2 \geq 4 \), we can build a two-dimensional data cube \( \langle C_\text{core}, S_\text{all} \rangle \) with dimensions \( D_1, D_2 \), such that \( C_\text{core} \) is non-trivially compromised by \( S_1 \).

**The Base Case:** When \( d_1 = d_2 = 4 \), the data cube shown in Table 4 validates the base case of our inductive hypothesis.

**The Inductive Case:** Assuming that there exists non-trivially compromisable two-dimensional data cube \( \langle C_\text{core}, S_\text{all} \rangle \) with dimensions \([1, d_1]\) and \([1, d_2]\), we show how to obtain a non-trivially compromisable two-dimensional data cube with dimensions \([1, d_1 + 1]\) and \([1, d_2]\).

Without loss of generality suppose that the tuple \((1, 1)\) is non-trivially compromised. Hence the RREF of \( M_1 \) is non-trivially compromised. Hence the tuple \((1, 1)\) in \( C_\text{core} \) is non-trivially compromised, validating the inductive case of our inductive hypothesis.

We briefly describe how to extend this result for \( k = 2 \) to \( k > 2 \). We do so by regarding part of the \( k \) dimensional data cube as a special two-dimensional data cube. Specifically, given \( k \) dimensional data cube \( \langle C_\text{core}, S_\text{all} \rangle \), let \( C_\text{core} = \{ t: t \in C_\text{core}, t[j] = 1 \forall 3 \leq j \leq k \} \) and \( C \) be the collection of all \( 1^\ast \) aggregation vectors satisfying \( \forall t \in C \forall j \in [3, k], t[j] = 1 \). We have \( Q \) set \( (C) = C_\text{core} \). Hence \( M_1 \) can be represented as:

\[
\begin{pmatrix}
  M_1' \mid 0 \\
  \end{pmatrix},
\]

where \( M_1' \) is a \(|C|\) by \(|C_\text{core}|\) sub-matrix and 0 is the \(|C|\) by \(|C_\text{core} \setminus C_\text{core}|\) zero matrix. \( M_1' \) can be treated as the \( 1^\ast \) aggregation matrix of a special two-dimensional data cube. We can build \( C_\text{core} \) in such a way that this two-dimensional data cube is non-trivially compromised. Hence the RREF of \( M_1' \) contains at least one unit row vector, which implies the RREF of \( M_1 \) does so, too. \( \square \)
We have two results for non-trivial compromises as stated in Theorem 2. The first says that full core cuboids cannot be non-trivially compromised. The first claim of Theorem 2 together with the first claim of Theorem 1 proves that any data cube with a full core cuboid is non-compromisable. In the proof of the first claim in Theorem 2, we construct a set of $2^k$ column vectors to contain any given column vector in the aggregation matrix. This set of $2^k$ vectors satisfies the property that any of its $2^k$ members can be represented as the linear combination of the other $2^k - 1$ members. The elementary row transformation used to obtain RREF of a matrix does not change the linear dependency of the column vectors. Hence the linear dependency among the $2^k$ column vectors also holds in the RREF of the aggregation matrix. Consequently the $2^k$ tuples corresponding to those columns cannot be non-trivially compromised.

The second and third claims of Theorem 2 give a tight lower bound on the cardinality of any core cuboid with missing tuples, such that it remains free of non-trivial compromises. The lower bound $2^{d_l} + 2^{d_m} - 9$ is a function of the two least dimension cardinalities. Intuitively, the justification of the lower bound is based on the following fact. The least number of missing tuples necessary for any non-trivial compromise increases monotonically with the number of aggregation cuboids involved in the compromise. This number reaches its lower bound when exactly two aggregation cuboids are used for non-trivial compromises. This is exactly the case given by the second claim of Theorem 2. The second claim shows that it is impossible to derive non-trivial compromisability criteria solely based on the cardinality of core cuboids, when the cardinality is not greater than the given lower bound. Thus the lower bound given by the third claim is the best possible.

**Theorem 2** (Non-trivial Compromisability).

1. In any given data cube $< C_{\text{core}}, S_{\text{all}} >$, $C_{\text{full}}$ cannot be non-trivially compromised by $S_{\text{all}}$.
2. For any integers $k$ and $d_1, d_2, \ldots, d_k$ satisfying $d_i \geq 4$ for all $1 \leq i \leq k$, let $d_l$ and $d_m$ be the least two among $d_i$s. Then there exists a $k$-dimensional data cube $< C_{\text{core}}, S_{\text{all}} >$ with $k$ dimensions $D_1, D_2, \ldots, D_k$, such that $|C_{\text{full}} \setminus C_{\text{core}}| = 2^{d_m} + 2^{d_m} - 9$ and $C_{\text{core}}$ is non-trivially compromised by $S_1$.
3. Given a $k$ dimensional data cube $< C_{\text{core}}, S_{\text{all}} >$ with dimensions $D_1, D_2, \ldots, D_k$, suppose $D_l$ and $D_m$ ($1 \leq l, m \leq k$) are the two with the least cardinalities. If $|C_{\text{full}} \setminus C_{\text{core}}| < 2|D_l| + 2|D_m| - 9$ holds, then $C_{\text{core}}$ cannot be non-trivially compromised.

**Proof.**

1. Due to the second claim of Lemma 1 we only need to shown that $C_{\text{full}}$ cannot be non-trivially compromised by $S_1$. Without loss of generality, we show that $t_0 = (1, 1, \ldots, 1)$ cannot be non-trivially compromised by $S_1$. In order to do so, we define $C'_{\text{full}} = \{ t: \forall i \in [1, k], t[i] = 1 \lor t[i] = 2 \}$. We then have $C'_{\text{full}} \subseteq C_{\text{full}}$ and $|C'_{\text{full}}| = 2^k$. Let $M'$ be a matrix comprising of the $2^k$ columns of $M_1$. 

that corresponds to $C_{\text{full}}$. In the rest of the proof we formally show that each of those $2^k$ column vectors can be represented as the linear combination of the rest $2^k - 1$ column vectors. It then follows from linear algebra that the RREF of $M_I$ does not contain $e_1$. Hence $t_0$ cannot be non-trivially compromised by $S_1$.

First we define the sign assignment vector as an $2^k$ dimensional column vector $t_{\text{sign}}$ as follows:

- $t_{\text{sign}}[1] = 1$
- $t_{\text{sign}}[2^i + j] = -t_{\text{sign}}[j]$ for all $0 \leq i \leq k - 1$ and $1 \leq j \leq 2^i$

**Claim:** $M'_t \cdot t_{\text{sign}} = 0$, where 0 is the $2^k$ dimensional zero column vector.

**Justification:**

Let $t_{\text{aggr}}$ be the $i^{th}$ 1-* aggregation vector and suppose $t_{\text{aggr}}[l] = *$ for some $1 \leq l \leq k$.

Let $r$ be the $i^{th}$ row of $M'_t$.

If $t_{\text{aggr}}[j] \leq 2$ for all $j \neq l$.

Then $|Q_{\text{set}}(t_{\text{aggr}}) \cap C_{\text{full}}'| = 2$, and suppose $Q_{\text{set}}(t_{\text{aggr}}) = \{t_1, t_2\}$

where $t_1 = C_{\text{full}}'[j_1]$ and $t_2 = C_{\text{full}}'[j_2]$, $t_1[l] = 1, t_2[l] = 2$

and $t_1[j] = t_2[j] = t_{\text{aggr}}[j]$ for all $j \neq l$

Hence, $j_1, j_2$ satisfy

$\forall j_1, j_2$ s.t. $r[j_1] = r[j_2] = 1$ and $r[j] = 0$ for any $j \neq j_1 \land j \neq j_2$.

The $j_1$th and $j_2$th columns of $M'_t$ correspond to $t_1$ and $t_2$ respectively.

Since $C_{\text{full}}'$ is in dictionary order, we have $j_2 = j_1 + 2^{l-1}$.

Hence, we have $r \cdot t_{\text{sign}} = 0$.

Otherwise, $|Q_{\text{set}}(t_{\text{aggr}}) \cap C_{\text{full}}'| = 0$.

Hence, $r = 0$ and $0 \cdot t_{\text{sign}} = 0$.

Hence, as stated earlier, the justification of our claim concludes the main proof.

2. Without loss of generality assume $m = 1$ and $n = 2$. Analogous to the proof for the third condition of Lemma 1, it suffices to consider only the case $k = 2$.

For an arbitrary but fixed value of $d_2$, we show by induction on $d_1$ that the data cube as constructed in the proof for the third condition of Lemma 1 satisfies $|C_{\text{full}} \setminus C_{\text{core}}| = 2d_1 + 2d_2 - 9$.

**The Inductive Hypothesis:** $C_{\text{core}}$ as constructed in the proof of Lemma 1 satisfies:

- $|C_{\text{full}} \setminus C_{\text{core}}| = 2d_1 + 2d_2 - 9$.
- $|P_1(C_{\text{full}}, d_1) \setminus P_1(C_{\text{core}}, d_1)| = 2$.

**The Base Case:** In Table 4, the core cuboid $C_{\text{core}}$ satisfies $|C_{\text{full}} \setminus C_{\text{core}}| = 2d_1 + 2d_2 - 9$. We also have $|P_1(C_{\text{full}}, 4) \setminus P_1(C_{\text{core}}, 4)| = 2$. This validates the base case of our inductive hypothesis.

**The Inductive Case:** Suppose we have two-dimensional data cube $C_{\text{core}}, S_{\text{full}} >$ with dimensions $D_1$ and $D_2$ satisfying $|C_{\text{full}} \setminus C_{\text{core}}| = 2d_1 + 2d_2 - 9$ and $|P_1(C_{\text{full}}, d_1) \setminus P_1(C_{\text{core}}, d_1)| = 2$. Use $C_{\text{core}}, S_{\text{full}} >$ and $C_{\text{full}}$ for the data
cube and full core cuboid with dimensions \([1, d_1 + 1]\) and \(D_2\), respectively. By the definition of \(C\) in the proof of Lemma 1 |\(C| = |P_1(C_{\text{core}}, d_1)|\), and as a consequence |\(C_{\text{full}} \setminus C_{\text{core}}\)| = |\(C_{\text{full}} \setminus C_{\text{core}}\)| + 2 = 2(d_1 + 1) + 2d_2 - 9. Since 
\(P_1(C_{\text{core}}, d_1 + 1) = C\), we have |\(P_1(C_{\text{full}}, d_1 + 1) - P_1(C_{\text{core}}, d_1 + 1)\)| = 2. This validates the inductive case of our inductive argument and consequently concludes our proof.

**Lower Bound:** Similarly we assume \(m = 1\) and \(n = 2\). We show that if \(C_{\text{core}}\) is non-trivially compromised then we have |\(C_{\text{full}} \setminus C_{\text{core}}\)| \(\geq 2d_1 + 2d_2 - 9\). First we make following assumptions.

(a) Tuple \(t = (1, 1, \ldots, 1) \in C_{\text{core}}\) is non-trivially compromised by \(S_1\).
(b) No tuple in \(C_{\text{core}}\) is trivially compromised by \(S_1\).
(c) There exists a minimal subset \(S\) of \(S_1\), such that for any \(C \in S\), \(t\) cannot be non-trivially compromised by \(S \setminus C\).
(d) For any \(t' \in C_{\text{full}} \setminus C_{\text{core}}\), \(t\) cannot be non-trivially compromised by \(S_1\) in data cube \(<C_{\text{core}}, \cup \{t'\}, S_{\text{full}}\>\). That is, |\(C_{\text{full}} \setminus C_{\text{core}}\)| reaches its lower bound.

Assumption 2 holds by Definition 4. Assumption 3 holds as by the first claim of Lemma 1, \(S\) must contain at least two \(1\)-\()^\ast\) aggregation cuboids. Assumption 4 holds because by Theorem 2, |\(C_{\text{full}} \setminus C_{\text{core}}\)| has a lower bound if \(C_{\text{core}}\) is non-trivially compromised.

**Claim:** Let \(C \in S\) and \(t[i] = \ast\) for any \(t \in C\). We have |\(P_i(C_{\text{full}}, 1) \setminus P_i(C_{\text{core}}, 1)\)| \(\geq 1\), and |\(P_i(C_{\text{full}}, j) \setminus P_i(C_{\text{core}}, j)\)| \(\geq 2\) for any \(2 \leq j \leq d_i\).

**Justification:** The proof is by contradiction. Without loss of generality, we only justify the claim for \(i = k\) and \(j = 2\). That is, given \(C \in S\) satisfying \(t[k] = \ast\) for any \(t \in C\) we prove that |\(P_i(C_{\text{full}}, 1) \setminus P_i(C_{\text{core}}, 1)\)| \(\geq 1\) and |\(P_i(C_{\text{full}}, 2) \setminus P_i(C_{\text{core}}, 2)\)| \(\geq 2\).

First we transform \(M_{C_{\text{core}}, S}\) into a singly bordered block diagonal form (SBBDF) [38] with row permutations, denoted by \(M_{m \times n}\). The \(i\)th diagonal block of \(M\) corresponds to \(P_k(C_{\text{core}}, i)\) and \(\{t: t \in S \setminus C \land t[k] = i\}\), and the border of \(M\) corresponds to \(C\). For example, Table 9 illustrates the SBBDF of \(M_{S_1, C_{\text{core}}}\) in Table 6. We call the columns of \(M\) containing the \(i\)th diagonal block as the \(i\)th slice of \(M\), also we use notation \(M[-, i]\) for the \(i\)th column of \(M\) and \(M[i, -]\) for the \(i\)th row.

Due to Assumption 1, there exists a \(m\)-dimensional row vector \(a\) satisfying \(a \cdot M = e_1\). Use \(r_i\) for \(M[i, -]\) then we get \(e_1 = \sum_{i=1}^{m} a[i] \cdot r_i\). Suppose each diagonal block of \(M\) has size \(m' \times n'\). Use \(r_i^{j}\), for \(1 \leq j \leq d_k\), to represent the sub-row vector of \(r_i\) intersected by the \(j\)th slice of \(M\). We have |\(r_i^{j}\)| = \(n'\). We also use \(e_1'\) and \(0'\) to represent the \(n'\) dimensional unit row vector and \(n'\) dimensional zero row vector, respectively. Then the following are true:

1. \(e_1' = \sum_{i=1}^{m'} a[i] r_i^{1} + \sum_{i=m'-m'+1}^{m} a[i] r_i^{1}\)
2. \(0' = \sum_{i=m' + 1}^{m} a[i] r_i^{2} + \sum_{i=m-m'+1}^{m} a[i] r_i^{2}\).
First we suppose \(|P_k(C_{full}, 2) \setminus P_h(C_{core}, 2)| = 0\), that is, the second slice of \(M\) contains no zero column. We then derive a contradiction to our assumptions.

Because \(|P_k(C_{full}, 2) \setminus P_h(C_{core}, 2)| = 0\), the first slice of \(M\) contains no less zero columns than the second slice of \(M\). Intuitively if the latter can be transformed into a zero row vector by some elementary row transformation, then applying the same transformation on the former leads to a zero vector, too. This is formally represented as:

\[
\text{iii. } 0' = \sum_{i=1}^{m'} a[m' + i]r_i + \sum_{i=m-m'+1}^{m} a[i]r_i.
\]

Subtracting (iii) from (i) gives \(e_1' = \sum_{i=1}^{m'} (a[i] - a[m' + i])r_i\). That implies \(C_{core}\) is non-trivially compromised by \(S - \{C_k\}\), contradicting Assumption 3.

Thus \(|P_k(C_{full}, 2) \setminus P_h(C_{core}, 2)| > 0\).

Next assume \(|P_h(C_{full}, 2) \setminus P_k(C_{core}, 2)| = 1\) and derive a contradiction to our assumptions.

Row vector \(r_3\) satisfies the following condition:

\[
\text{iv. } 0' = \sum_{i=2}^{m} a[i]r_i^1 + \sum_{i=m-m'+1}^{m} a[i]r_i^3.
\]

Let \(t' \in P_k(C_{full}, 2) \setminus P_h(C_{core}, 2)\). Notice that (i), (ii) still hold. Suppose \(t'\) corresponds to \(M[-, y]\), which is a zero column. Now assume that we add \(t'\) to \(P_h(C_{core}, 2)\). Consequently we have that \(M[-, y] \neq 0\). Due to Assumption 4, the left side of (ii) must now become \(e_1'\). That is, \(a \cdot M[-, y] = 1\). There will also be an extra 1-element \(M[x, y]\) in the border of \(M\).

Now let \(t''\) be the tuple corresponding to \(M[-, y + n']\) in the third slice of \(M\). Suppose \(t'' \in P_k(C_{core}, 3)\) and consequently \(M[-, y + n'] \neq 0\). We have that \(M[-, y + n'] = M[-, y] + 1\) and consequently \(a \cdot M[-, y + n'] = 1\).

Now removing \(t'\) from \(P_k(C_{core}, 2)\). Now we show by contradiction that \(t'' \in P_k(C_{core}, 3)\) cannot be true. Intuitively, because \(t'\) is the only missing tuple in the second slice of \(M\), the third slice of \(M\) contains no less zero vectors than the second slice of \(M\) does, except \(t''\). Because \(a \cdot M[-, y + n'] = 1\), the elements of the vector \(a\), which transforms the second slice of \(M\) to a zero
vector as shown by (ii), can also transform the third slice of $M$ to a unit vector. This is formally represented in (v):

$$v = e'' = \sum_{i=2m'+1}^{3m'} a[i-m']^2 + \sum_{i=m-m'+1}^m a[i]^2.$$  

Subtracting (iv) from (v) we get $e'' = \sum_{i=2m'+1}^{3m'} (a[i-m'] - a[i])^2$, implying $C_{core}$ is compromised by $S \setminus \{C_i\}$. Hence, Assumption 3 is false. Consequently, $t'' \not\in C_{core}$.

A similar proof exists for the $i$th slice of $C_e$ for any $4 \leq i \leq d_k$. However, $M[x, -] \neq 0$ because otherwise we can let $\alpha_x$ be zero and then decrease $|C_{full} \setminus C_{core}|$, contradicting Assumption 4. Hence $M[x, -]$ is a unit vector with the 1-element in the first slice of $M$. However, this further contradicts Assumption 2. That is, no trivial compromise is assumed. Hence we have that $|P_k(C_{full}, 2) \setminus P_k(C_{core}, 2)| = 1$ is false.

Now consider $|P_k(C_{full}, 1) \setminus P_k(C_{core}, 1)| = 0$. Let $t_1, t_2 \in P_k(C_{full}, 2) \setminus P_k(C_{core}, 2)$. Now define $C''_{core} = C_{core} \setminus \{t\} \cup \{t_1\}$, and use $M'$ for $M_{C''_{core}}$. From $a \cdot M = e_1$ and Assumption 4 we get that $a \cdot M' = e_1$, and $M[-, i]$ corresponds to $t_1$. This implies that $t_1$ is non-trivially compromised in $<C''_{core}, S_{full}>$, with $|P_k(C_{full}, 1) \setminus P_k(C''_{core}, 1)| = 1$, which contradicts what we have already proved. Hence, we get $|P_k(C_{full}, 1) \setminus P_k(C_{core}, 1)| \geq 1$. This concludes the justification of our claim.

The justified claim implies that the number of missing tuples in $C_{core}$ increases monotonically with the following:

- The number of aggregation cuboids in $S$.
- $d_i$, if there exists $C \in S$ satisfying $t[i] = *$ for any $t \in C$.

Hence $|C_{full} \setminus C_{core}|$ reaches its lower bound when $|S| = 2$, which is equal to $2D_1 + 2D_2 - 9$, as shown in the first part of the current proof - concluding the proof of Theorem 2. 

The results stated in Corollary 1 follow from Theorem 2 and Lemma 1 but have value of their own. The first claim of Corollary 1 says that if the $i$th 1-* aggregation cuboid is essential for any non-trivially compromises, then every slice of the core cuboid on the $i$th dimension must contain at least one missing tuples. As an example, in the core cuboid shown in Table 4, every slice on the two dimensions contains either one or two missing tuples. The second claim of Corollary 1 says that any core cuboid that have full slices on $k - 1$ of the $k$ dimensions must be safe from non-trivial compromises.

**Corollary 1** (Non-trivial Compromisability With Full Slices). In any data cube $<C_{core}, S_{full}>$ with dimensions $D_1, D_2, \ldots, D_k$: 

- The number of aggregation cuboids in $S$.
- $d_i$, if there exists $C \in S$ satisfying $t[i] = *$ for any $t \in C$.
1. Let $C \in S \subseteq S_1$ and suppose $t[i] = \ast$ for any $t \in C$, where $i \in [1, k]$ is fixed. If $S$ non-trivially compromises $C_{\text{core}}$ but $S \setminus \{C\}$ does not, then $|P_i(C_{\text{full}}, j) \setminus P_i(C_{\text{core}}, j)| \geq 1$ for any $1 \leq j \leq d_i$.

2. If there exists $S \subseteq [1, k]$ satisfying $|S| = k - 1$ and for any $i \in S$, there exists $j \in D_i$ such that $|P_i(C_{\text{full}}, j) \setminus P_i(C_{\text{core}}, j)| = 0$, then $C_{\text{core}}$ cannot be non-trivially compromised by $S_{\text{full}}$.

**Proof.** The first claim follows directly from the proof of Theorem 2. The second claim then follows from the first claim of Corollary 1 taken together with the first claim of Lemma 1.

5. Three-tier inference control model

Traditional view of inference control has two tiers, that is, the data set and the answerable queries. The data set is usually modeled as a set of tuples, similar to the core cuboid defined in Section 3. A query selects a subset of tuples in the data set, called the query set of the query. The query then aggregates (for example, sums) the values of those tuples in the query set. Inference becomes a concern when the values being aggregated by the query are sensitive. With this two tier view, a typical restriction-based inference control mechanism checks a given set of queries for unwanted inferences and answers only those queries that do not compromise the sensitive values.

Inference control based on the two tier view has some inherent drawbacks. First of all, allowing ad-hoc queries unnecessarily complicates inference control. Because any subset of a given data set may potentially be the query set of a query, a total of $2^n$ queries with different query sets are possible on a data set containing $n$ vectors.
Second of all, inferences can be obtained with a single query as well as by manipulating multiple queries, as shown by the third and fourth quarter data in Table 1, respectively. Hence totally $2^{2n}$ different sets of queries can be formed on the data set. Such a large number of possibilities partially contributes to the high complexity of most existing inference control mechanisms. In practice most ad-hoc queries are either meaningless to users or redundant because their results can be derived from other previously answered queries. For example, for SUM queries, at most $n$ queries with different query sets can be formed on a data set of size $n$ before the result of any new query can be derived from the old ones, because the rank of an aggregation matrix with $n$ columns cannot exceed $n$.

Inference control mechanisms developed under the two tier view usually have high on-line computational complexity. Here on-line computations refer to the computations conducted after queries have been posed to the system, and conversely off-line computations occur before queries are posed. In the two tier view of inference control, it is difficult for restriction-based inference control mechanisms to predict how incoming queries will aggregate data. Hence the major part of computational effort required by inference control mechanisms cannot be started until queries are received. Consequently, the time used by inference control adds to the system response time. This is unacceptable considering the high complexity of most inference control mechanisms.

Finally, rich dimension hierarchies embedded in most multi-dimensional data sets are ignored by the two tier view of inference control. That prevents inference control mechanisms from benefiting from those hierarchies. In Table 1, the dimension hierarchy composing of month, quarter and year naturally divides the data set into four blocks, shown as the four cross-tabulations. In OLAP systems, most meaningful queries are formed on the basis of those partitions. As an example, a query that sums Alice’s commission in January and Bob’s commission in August conveys little useful information to users. Without taking dimension hierarchies into account, inference control mechanisms have to take the whole data set as their input, even when queries involve only a block of the data set.

To address the listed issues, we propose a three-tier inference control model consisting of three tiers with three relations in between, as shown in Fig. 1. The data tier $D$ is a set of tuples. Both aggregation tier $A$ and query tier $Q$ are sets of queries. We do not consider the details of tuples and queries here, but instead we consider them as elements of sets. The relation $R_{QD}$ is defined as the composition of $R_{QA}$ and $R_{AD}$. We assume that a suitable definition of compromisability has been given. In addition we enforce three properties on the model as the follows.

1. **Three Tiers:**
   a. Data Tier $D$.
   b. Aggregation Tier $A$.
   c. Query Tier $Q$. 
2. Relations Between Tiers:
(a) $R_{AD} \subseteq A \times D$.
(b) $R_{QA} \subseteq Q \times A$.
(c) $R_{QD} = R_{AD} \circ R_{QA}$.

3. Properties:
(a) $|A|$ is polynomial in $|D|$.
(b) $D$ and $A$ can be partitioned into $D_1, D_2, \ldots, D_m$ and $A_1, A_2, \ldots, A_m$, satisfying that $(a, d) \in R_{AD}$ only if $d \in D_i$ and $a \in A_i$ for some $1 \leq i \leq m$.
(c) $A$ does not compromise $D$.

Proposition 1 explains how the three-tier model controls inferences. Intuitively, the aggregation set $A$ provides a set of intermediate aggregations that are guaranteed to be safe from compromises. Queries in $Q$ are then answered by deriving results from these intermediate aggregations. The query results will not compromise any tuple in $D$ because they do not convey any information beyond those contained in the aggregation tier $A$.

**Proposition 1.** The three-tier model guarantees that $Q$ does not compromise $D$.

**Proof.** For any given set of queries $Q' \subseteq Q$, we have that $R_{QD}(Q') = R_{AD}(R_{QA}(Q'))$ because $R_{QD} = R_{AD} \circ R_{QA}$. Hence $Q'$ does not compromises $D$ if the aggregations given by $R_{QA}(Q')$ do not compromises $D$. We have that $R_{QA}(Q') \subseteq A$ holds by definition. The third property of the model then guarantees that $R_{QA}(Q')$ does not compromise $D$. □

In contrast to traditional two tier view of inference control, the three-tier model improves the performance of inference control mechanisms in several ways. Firstly,
the size of the input to inference control mechanisms is dramatically reduced. Different from the two tier view, the three-tier view makes aggregation tier $A$ the input of inference control mechanisms. The third property of the three-tier model requires an aggregation tier $A$ to have a size comparable to that of the data tier $D$. Choosing such an aggregation tier $A$ is possible, because as we have explained, the number of non-redundant aggregations is bounded by the size of the data tier $D$.

Secondly, the three-tier model facilitates using a divide-and-conquer approach to further reduce the size of inputs to inference control mechanisms by localizing the inference control problem. Due to the second property of the model, for any $1 \leq i \leq m$ we have that $R_{AD}(A_i) \subseteq D_i$ and $R_{AD}(D_i) \subseteq A_i$. Hence for any given set of queries $Q' \subseteq Q$ satisfying $R_{QA}(Q') \subseteq A_i$ for some $1 \leq i \leq m$, the compromisability depends on $A_i$ and $D_i$ only. Intuitively, if a set of queries can be answered with some blocks of the aggregation tier $A$, then the inference control problem of the queries can be localized to these blocks of $A$ and their $R_{AD}$-related blocks of the data tier $D$.

In practice, partitioning $D$ and $A$ to satisfy the second property is possible because most multi-dimensional data sets contain dimension hierarchies.

Finally, the three-tier model shifts the major computational effort required by inference control to off-line processing, thereby reduces on-line performance cost. Composing aggregation tier $A$ by defining $R_{AD}$ to satisfy the three properties of the model is the most computationally expensive task. This task can be processed off-line, before answering any queries. The on-line processing consists of decomposing any given set of queries $Q' \subseteq Q$ by computing $R_{QA}(Q')$. Because most OLAP systems utilize pre-computed aggregations for query-answering, this query decomposition mechanism is in place or can be implemented easily. Hence by decoupling off-line and on-line processing of inference control and pushing computational complexity to the off-line part, the three-tier model can eliminate or reduce the delay of query-answering caused by inference control.

Defining $R_{QD}$ as the composition of $R_{AD}$ and $R_{QA}$ may reduce the total number of answerable queries. This restriction reflects the unavoidable trade-off between availability and security. However, the design of aggregation tier $A$ should enable it to convey as much useful information as possible, while not endangering the sensitive information stored in the data tier $D$. The usefulness of queries usually depends on application settings. For example, in OLAP applications data cube style aggregations, as modeled in Section 3, are the most popular queries users may pose to the system.

### 6. Cardinality-based inference control for data cubes

In this section we describe an inference control algorithm that integrates the cardinality-based compromisability criteria developed in Section 4. We then show its correctness and computational complexity. Finally we discuss implementation issues and improvements to the algorithm.
6.1. Inference control algorithm

Our inference control algorithm is based on the three-tier inference control model discussed in Section 5. According to the model, inference control resides between the data tier and aggregation tier. Hence we shall focus on those two tiers. We briefly address the query tier in Section 6.3.

Given a data cube \(<C_{\text{core}}, S_{\text{all}}>\), the core cuboid \(C_{\text{core}}\) constitutes the data tier. A collection of aggregation vectors \(S_A\), to be defined shortly, constitutes the aggregation tier. A tuple \(t \in C_{\text{core}}\) is \(R_{\text{AD}}\)-related to an aggregation vector \(t_{\text{aggr}} \in S_A\) if and only if \(t \in \text{Qset}(t_{\text{aggr}})\).

The definition of \(S_A\) is based on the following partition on \(C_{\text{core}}\). For \(i = 1, 2, \ldots, k\), we divide the \(i\)th dimension \([1, d_i]\) into integer intervals \([1, d_1], [d_1 + 1, d_2], \ldots, [d_i, d_{i+1}+1, d_i]\), where \(m_i\) is fixed for each \(i\). Then we partition \(C_{\text{core}}\) into \(\prod_{i=1}^k m_i\) blocks using those intervals. The partition of \(C_{\text{core}}\) satisfies the property that any two tuples \(t, t' \in C_{\text{core}}\) are in the same block if and only if their \(i\)th elements are in the same block of the \(i\)th dimension for all \(i = 1, 2, \ldots, k\). In the partition of \(C_{\text{core}}\), we regard each block, denoted as \(C_{s_{\text{core}}}^s\), as the core cuboid of a sub-data cube \(<C_{s_{\text{core}}}^s, S_{s_{\text{all}}}^s>\) (from now on we use symbols with superscripts for sub-data cubes and their components). \(S_A\) is the collection of \(1^*\) aggregation vectors with non-empty aggregation sets in all the safe sub-data cubes. Notice that when integrating the aggregation vectors defined in sub-data cubes into \(S_A\), the aggregation set of each aggregation vector remains the same as it is originally defined in the sub-data cube. For example, the data cube in Table 1 is partitioned into four sub-data cubes, each represented by a cross-tabulation. The subtotals shown in the table correspond to the \(1^*\) aggregation vectors defined in sub-data cubes, and therefore constitute the members of \(S_A\).

The inference control algorithm \(Ctrl_{\text{Inf Cube}}\) shown in Fig. 2 constructs the aggregation set tier \(S_A\) by partitioning the core cuboid \(C_{\text{core}}\). The main routine of \(Ctrl_{\text{Inf Cube}}\) accepts as input a given data cube \(<C_{\text{core}}, S_{\text{all}}>\) and its \(k\) dimensions \(D_1, D_2, \ldots, D_k\). For \(i = 1, 2, \ldots, k\), a set of \(m_i - 1\) integers between 1 and \(d_i\) partition the \(i\)th dimension \([1, D_i]\) into \(m_i\) blocks, and consequently partition \(C_{\text{core}}\) into \(\prod_{i=1}^k m_i\) blocks. Each block of \(C_{\text{core}}\) is then normalized to be a core cuboid \(C_{s_{\text{core}}}^s\) as stated in Definition 1, by converting its \(k\) dimensions to integer intervals starting from 1. The core cuboid \(C_{s_{\text{core}}}^s\) is then passed to the subroutine \(Ctrl_{\text{Inf Sub}}\). The subroutine \(Ctrl_{\text{Inf Sub}}\) applies the cardinality-based sufficient conditions given in Section 4 to the sub-data cube \(<C_{s_{\text{core}}}^s, S_{s_{\text{all}}}^s>\), in order to determine its compromisability. The \(1^*\) aggregation vectors with non-empty aggregation sets in those non-compromisable sub-data cubes are returned to the main routine, and collectively constitute the aggregation tier.

As an example, suppose we feed a core cuboid similar to Table 1 into the algorithm \(Ctrl_{\text{Inf Cube}}\), with each quarter as a block. The subroutine \(Ctrl_{\text{Inf Sub}}\) conducts five tests consecutively for each such block. Because the first block corresponds to a full core cuboid, the second test succeeds. Hence the subroutine returns all the \(1^*\)
### Algorithm Ctrl_Inf_Cube

**Input:** data cube \(< C_{core}, S_{all} >\) with dimensions \(D_1, D_2, \ldots, D_k\), and integers 
\(1 < d^1_i < d^2_i < \cdots < d^{m_i - 1}_i < d_i\) for \(1 \leq i \leq k\), where \(m_i\) is fixed for each \(i\) and let \(d^0_i = 0, d^{m_i}_i = d_i\)

**Output:** a set of aggregation vectors \(S_A\)

**Method:**
1. Let \(S_A = \phi\);
2. For each \(k\) dimensional vector \(v\) in vector space \(\prod_{i=1}^{k} [1, m_i]\)
   - Let \(C_{tmp} = \{ t: t \in C_{core}, \forall i \in [1, k] \ t[i] \in [d^i_t]_i - 1, d^i_t[i] \}\);
   - Let \(C_{core} = \{ t: \exists t' \in C_{tmp}, \forall i \in [1, k] \ t[i] = t'[i] - d^i_{t'[i]} - 1 \}\); 
   - Let \(S_A = Ctrl_Inf_Sub( C_{core} )\);
   - Let \(S_A = S_A \cup S_A\);
3. Return \(S_A\).

### Subroutine Ctrl_Inf_Sub

**Input:** \(k\) dimensional core cuboid \(C^s_{core}\)

**Output:** a set of \(1-*\) aggregation vectors if \(S^s_1\) does not compromise \(C^s_{core}\), and \(\phi\) otherwise

**Method:**
1. If \(|C^s_{core}| < 2^{k-1} \cdot \text{max}(d^s_1, d^s_2, \ldots, d^s_k)\) 
   Return \(\phi\);
2. If \(|C^s_{core}| = |C^s_{full}|\)
   Return \(\cup_{C \in S_1} C\);
3. If \(C^s_{core}\) is trivially compromised by \(S^s_1\)
   Return \(\phi\);
4. If \(|C^s_{full} - C^s_{core}| < 2d^s_i + 2d^s_m - 9\), where \(d^s_i, d^s_m\) are the two smallest among \(d^s_i\)s
   Return \(\cup_{C \in S_1} C\);
5. If there exists \(D \subset [1, k]\) satisfying \(|D| = k - 1\) and for any \(i \in D\), 
   there exists \(j \in [1, d^s_i]\) 
   such that \(|P_i(C^s_{full}, j) \setminus P_i(C^s_{core}, j)| = 0\)
   Return \(\cup_{C \in S_1} C\);
6. Return \(\phi\).

Fig. 2. The algorithm of inference control in data cube.

aggregation vectors as its output. The second block has full slices on both dimensions, and hence the fifth test for full slices succeeds with all the \(1-*\) aggregation vectors returned. The third block is determined by the third test as trivially compromised, so nothing is returned for this block. Finally the fourth block fails all the five tests and nothing is returned.
6.2. Correctness and time complexity

Now we prove the correctness of algorithm Ctrl_Inf_CUBE. More specifically, we show that the aggregation tier constructed by the algorithm satisfies the first and second properties of the model discussed in Section 5. The last property holds trivially. The first property, namely, \(|S_A|\) being a polynomial in \(|C_{core}|\) is justified in Proposition 2.

**Proposition 2.** \(S_A\) constructed in the algorithm Ctrl_Inf_CUBE satisfies: \(|S_A| = O(|C_{core}|)\).

**Proof.** Let \(n = |C_{core}|\). The 1-* aggregation matrix \(M_1^*\) of any sub-data cube \(<C_{core}, S_{all}>\) has \(n^* = |C_{core}^*|\) non-zero columns. Hence \(M_1^*\) has \((n^* \cdot k)\) many 1-elements, implying at most \((n^* \cdot k)\) non-zero rows in \(M_1^*\). Hence the set of 1-* aggregation vectors with non-empty aggregation sets has a cardinality no bigger than \((n^* \cdot k)\), that is, \(|S_A^*| \leq n^* \cdot k\). Consequently, we have \(|S_A| \leq k \cdot \sum n^* = O(n)|, assuming \(k\) is constant compared to \(n\). \(\square\)

The second property of the model, namely, the ability to partition \(S_A\) and \(C_{core}\) is satisfied by the partitions constructed in the algorithm Ctrl_Inf_CUBE. Let \(t^*_\text{aggr}\) be any aggregation vector in sub-data cube \(<C_{core}^*, S_{all}^*>\). We have \(Q(\text{set}(t^*_\text{aggr})) \subseteq C_{core}^*\). Because as stated earlier, in the data cube \(<C_{core}, S_{all}>, Q(\text{set}(t^*_\text{aggr}))\) remains the same as it is defined in \(<C_{core}^*, S_{all}^*>\). Hence it is also a subset of \(C_{core}^*\). Consequently, a tuple \(t \in C_{core}\) is in \(Q(\text{set}(t^*_\text{aggr}))\) only if \(t \in C_{core}^*\). That is, \(t\) and \(t^*_\text{aggr}\) are \(R_{AD}\) related only if they are in the same sub-data cube \(<C_{core}^*, S_{all}^*>\).

**Proposition 3.** The computational complexity of the algorithm Ctrl_Inf_CUBE is \(O(|C_{core}|)\).

**Proof.** Let \(n = |C_{core}|\). The main routine Ctrl_Inf_CUBE partitions \(C_{core}\) by evaluating the \(k\) elements of each tuple and assigning the tuple to the appropriate block of the partition. This operation has a runtime of \(O(nk) = O(n)\). The subroutine Ctrl_Inf_Sub is called for each of the \(\prod_{i=1}^{k} m_i\) blocks of the partition. For any block with cardinality \(n^*\), we show the complexity of the subroutine to be \(O(n^*)\).

First consider the matrix \(M_1^*\) that contains all non-zero rows and zero columns in \(M_1^*\). From the proof of Proposition 2, \(M_1^*\) has \(n^*\) columns, \(O(n^*)\) rows and \((n^* \cdot k)\) many 1-elements. Hence \(M_1^*\) can be populated in \(O(n^*)\) time (we only need to populate the 1-elements).

Once \(M_1^*\) is established, we show that the cardinality-based tests in the subroutine Ctrl_Inf_Sub can be done in \(O(n^*)\) time. The first, second and fourth tests can be done in constant time because they only need the cardinalities \(|C_{core}^*| = n^*\) and \(|S_{all}^*| = \prod_{i=1}^{k} d_i^*\). The third test for trivial compromisability can be done by counting the 1-elements of \(M_1^*\) in each row. The complexity is \(O(n^*)\) because the total number of 1-* elements is \((n^* \cdot k)\). The last test for full slices requires counting the number
of columns of $M'_1$ in each slice along all $k$ dimensions. Hence the complexity is $k \cdot n^s = O(n^s)$. □

The computational complexity of the algorithm, as stated in Proposition 3, is $O(n)$, where $n$ is the cardinality of the core cuboid $C_{core}$. On the other hand, determining compromisability by transforming an $m$ by $n$ aggregation matrix to its RREF has a complexity of $O(m^2n)$, and the maximum subset of non-compromisable aggregation vectors cannot be found in polynomial time [13]. Moreover, the complexity of our algorithm is handled by off-line processing, according to the three-tier inference control model.

6.3. Implementation issues

6.3.1. Integrating inference control into OLAP

Using pre-computed aggregations to reduce the response time in answering queries is a common approach in OLAP [19,27]. OLAP queries are split into sub-queries whose results are computed from cached aggregations. Our algorithm also uses pre-computed aggregations, only for a different purpose: inference control. It is a natural choice to integrate the inference control algorithm with caching mechanisms that already exist in OLAP applications. However, inference control may require modifications to such mechanisms. For example, after splitting a query into sub-queries, one may find that some sub-queries do not correspond to any aggregation in the aggregation tier. A caching mechanism may choose to answer them by computing them from the raw data. However, answering such sub-queries may lead to inferences because the disclosed information goes beyond the aggregation tier. In such a case the query should either be denied or modified in order for them to be answerable from the aggregation tier. For example, in Table 1, suppose that a query asks for the total commission of each employee in the first two quarters. This query can be split into two sub-queries corresponding to the subtotals in the first two quarters. The query is safe to answer, because all the required subtotals are in the aggregation tier. Next, suppose another query asks for the total commission of each employee in the first four months. Splitting this query leads to unanswerable sub-queries, because only the data of April is selected by the query. Answering this query discloses all individual commissions in April if the subtotals in the first quarter are previously answered.

The partitions on the data tier and the aggregation tier has an important impact on the usability of OLAP systems with inference control enforced. Partitioning should be based on dimension hierarchies, so that most useful queries correspond to whole blocks in the partitions. The choice of dimension granularity of the partitioning is also important. Most practical data sets have deep dimension hierarchies composing of many different granularities for each dimension, with the coarser ones at the top of the hierarchy and finer ones at the bottom. Choosing coarser dimensions as the basis for partitioning leads to fewer and larger blocks. Larger blocks cause less answerable queries. In the above example the query for commissions in the first four months
cannot be answered because the granularity used in the query (month) is finer than
the dimension used to partition the data (quarter). This can be avoided if blocks are
formed by months. However, such partitioning does not provide inference control at
all. Because of such subtleties, it may be attractive to use a query-driven or dynamic
partition. However, varying partitions at run-time causes more online performance
overhead and hence is less feasible.

6.3.2. Re-ordering tuples in unordered dimensions

In practice, many data cubes have unordered dimensions. That is, the order of
values in the domain of those dimensions have no apparent semantics associated
with it. For example, in Table 1 the dimension employee has no natural ordering.
In the core cuboid of a data cube tuples can usually be re-ordered such that their
orders in ordered dimensions are not affected. For example, in Table 1 assuming
the dimension employee is unordered, tuples can be horizontally re-ordered along
employee dimension.

Cardinality-based compromisability of data cubes depends on the density of each
block of the core cuboid. As shown in Section 4, full blocks or dense blocks with
cardinalities above the upper bound given in Theorem 2 cannot be non-trivially com-
promised. Without losing any useful information, the tuples in a core cuboid can be
re-ordered such that partitioning the core cuboid leads to more full blocks and dense
blocks. One consequence of this re-ordering is that aggregation tier will contain more
safe aggregations leading to better usability of the system.

Techniques already exist in increasing the number of dense blocks in data cubes
by re-ordering tuples along un-ordered dimensions. For example, the row shuffling
algorithm presented in [4] re-orders tuples in the core cuboid, so that the tuples con-
taining similar values are moved closer to each other. We can implement the row
shuffling algorithm as a step prior to the algorithm Ctrl_Inf_Cube, by applying it to
the full core cuboid of data cubes. In [4] the similarity between two tuples is de-

6.3.3. Update operations

Although update operations are less common in OLAP systems than they are in
traditional databases, the data in data warehouses need to be modified over time. Typ-
ical update operations include inserting or deleting tuples, and modifying the values
contained in tuples. Those updates need to be done efficiently to reduce their im-
pact on availability of the system. Three-tier inference control facilitates pre-defined
aggregations, which may also need to be updated as underlying data change.

Cardinality-based inference control is independent of the sensitive values. Hence
modifying sensitive values usually has no effect on compromisability and can be ig-
Subroutine Ctrl_Inf_Insert
Input: tuple \( t \in C_{core} \) to be inserted
Output: a set of aggregation vectors \( S_A \)
Method:
1. If \( t \) should be inserted into the sub-data cube \( C_{s_{core}}^n \) and \( S_A^n = \phi \)
   Let \( S_A = S_A \setminus S_A^n \);
   Let \( S_A^n = \text{Ctrl}_\text{Inf}_\text{Sub}_\text{Insert}( C_{s_{core}}^n, t ) \);
   Let \( S_{A} = S_A \cup S_A^n \);
3. Let \( C_{core} = C_{core} \cup \{ t \} \);
4. Return \( S_A \).

Subroutine Ctrl_Inf_Sub_Insert
Input: tuple \( t \) and sub-data cube \( C_{s_{core}}^n \)
Output: a set of aggregation vectors if \( S_I^n \) does not compromise \( C_{s_{core}}^n \cup \{ t \} \), and \( \phi \) otherwise
Method:
1. For each \( i \in [1, k] \)
   If \( t[i] \notin [1, d_i^n] \)
   \( d_i^n = d_i^n + 1 \);
2. If \( |C_{s_{core}}^n| + 1 < 2^{k-1} \cdot \max(d_1^n, d_2^n, \ldots, d_k^n) \)
   Return \( \phi \);
3. If \( C_{s_{core}}^n \cup \{ t \} \) is trivially compromised by \( S_I^n \)
   Return \( \phi \);
4. If \( |C_{full}^n - C_{core}^n| < 2d_l^n + 2d_m^n - 8 \), where \( d_l^n, d_m^n \) are the two smallest among \( d_i^n \)'s
   Return \( \cup_{C \in S_I^n} C \);
5. If there exists \( D \subset [1, k] \) satisfying \( |D| = k - 1 \) and for any \( i \in D \),
   there exists \( j \in [1, d_i^n] \)
   such that \( |P_i(C_{full}^n, j) \cup P_i(C_{core}^n, j)| = 0 \)
   Return \( \cup_{C \in S_I^n} C \);
6. Return \( \phi \).

Fig. 3. Insertion of a tuple.

ored by inference control mechanisms. For example, changing the commissions in Table 1 does not affect the compromisability of the data cube. On the other hand, modifying the non-sensitive values contained in a tuple may affect the compromisability, because the modified tuple may belong to a different block than the original tuple in the partition of the core cuboid. We treat the modification of non-sensitive values contained in a tuple as two separate operations, deletion of the original tuples and insertion of new tuples containing the modified values.

Figure 3 and Fig. 4 show the algorithms for the deletion and insertion of a tuple respectively. These two update operations are handled similarly, therefore we discuss
Subroutine **Ctrl_Inf_Delete**

**Input:** tuple \( t \in C_{\text{core}} \) to be deleted

**Output:** a set of aggregation vectors \( S_A \)

**Method:**

1. Find the sub-data cube \( C_{\text{core}}^s \) containing \( t \);
2. If \( S_A^s \neq \phi \)
   - Let \( S_A = S_A \setminus S_A^s \);
   - Let \( S_A^s = \text{Ctrl}_\text{Inf}_\text{Sub}_\text{Delete}(C_{\text{core}}^s, t) \);
3. Let \( C_{\text{core}} = C_{\text{core}} \setminus \{ t \} \);
4. Return \( S_A \).

Subroutine **Ctrl_Inf_Sub_Delete**

**Input:** tuple \( t \) and sub-data cube \( C_{\text{core}}^s \)

**Output:** a set of aggregation vectors if \( S_1^s \) does not compromise \( C_{\text{core}}^s \setminus \{ t \} \), and \( \phi \) otherwise

**Method:**

1. For each \( i \in [1, k] \)
   - If \( |P_i(C_{\text{core}}^s, \{i\})| = 1 \)
     - \( d_i^s = d_i^s - 1; \)
2. If \( |C_{\text{core}}^s| - 1 < 2^{k-1} \cdot \max(d_1^s, d_2^s, \ldots, d_k^s) \)
   - Return \( \phi \);
3. If \( C_{\text{core}}^s \setminus \{ t \} \) is trivially compromised by \( S_1^s \)
   - Return \( \phi \);
4. If \( |C_{\text{full}} - C_{\text{core}}|^s < 2d_i^s + 2d_m^s - 8 \), where \( d_i^s, d_m^s \) are the two smallest among \( d_i^s \)’s
   - Return \( \bigcup_{C \in S_1^s} C \);
5. If there exists \( D \subset [1, k] \) satisfying \( |D| = k - 1 \) and for any \( i \in D \), there exists \( j \in [1, d_i^s] \) such that \( |P_i(C_{\text{full}}^s, j) \setminus P_i(C_{\text{core}}^s, j)| = 0 \)
   - Return \( \bigcup_{C \in S_1^s} C \);
6. Return \( \phi \).

Fig. 4. Deletion of a tuple.

deletion only. The subroutine **Ctrl_Inf_Delete** in Fig. 4 updates \( S_A \) upon the deletion of a tuple. It first finds the sub-data cube that contains the tuple to be deleted. If the sub-data cube is already compromised before the deletion, then it must remain so after the deletion. Hence in such a case the subroutine **Ctrl_Inf_Delete** returns immediately. If the sub-data cube is not compromised before the deletion, the subroutine **Ctrl_Inf_Sub_Delete** is called to determine the compromisability of the sub-data cube after the deletion of the tuple. The subroutine **Ctrl_Inf_Sub_Delete** reduces a dimension cardinality by one if the corresponding value is contained in
the deleted tuple only. The cardinality-based compromisability criteria are then applied to the sub-data cube similarly as in Ctrl_Inf_Sub. There is no need to check if the core cuboid is full after deleting a tuple from it. The complexity of the subroutine Ctrl_Inf_Sub_Delete is bound by $O(|C_{core}|)$. The complexity can be reduced if we keep the cardinality results computed in the subroutine Ctrl_Inf_Sub and Ctrl_Inf_Sub_Delete, such as $|C_{core}|$ and $|P_1(C_{core},j)|$. Although more space is needed to do so, the subroutine Ctrl_Inf_Sub_Delete runs in constant time if those results do not need to be re-computed.

6.3.4. Aggregation operators other than sum

Although sum queries compose an important portion of OLAP queries, other aggregation operators such as count, average, max and min are also used in practice. In some applications counts also need to be protected from compromises. Count queries can be treated as sum queries on binary values, when considering the value of each tuple as one, and the value of each missing tuple as zero. It has been shown in [29] that compromisability of queries on binary values has a much higher complexity than the case of real numbers. Hence efficiently controlling inference of counting queries is still an open problem. Because we do not restrict counting queries, inference control of the queries that contain averages becomes equivalent to that of sums. Other aggregation operators exhibiting similar algebraic property with sum may also be handled by our algorithm. Holistic aggregation operators such as median invalidates the partitioning approach adopted by the three-tier model, because they cannot be calculated with partial results obtained in each block of the partition. Our on-going work addresses these issues.

7. Conclusions

We have derived sufficient conditions for sum-only data cubes safe from inferences. We have shown that compromisability of multi-dimensional aggregations can be reduced to those of one dimensional aggregations. Using the results, we have shown that core cuboids with no values known from external knowledge are free from inferences, and that there is a tight bound on the number of such know values for the data cube to remain safe from inferences. To apply our results for inference control of data cube queries, we have shown an inference control algorithm based on a three-tier model. Our ongoing work addresses aggregation operations other than sums.

Acknowledgements

The authors are grateful to the anonymous reviewers for their valuable comments.
References


