Efficient Computation of Iceberg Cubes by Bounding Aggregate Functions

Xiuzhen Zhang, Pauline Lienhua Chou, and Guozhu Dong, Senior Member, IEEE

Abstract—The iceberg cubing problem is to compute the multidimensional group-by partitions that satisfy given aggregation constraints. Pruning unproductive computation for iceberg cubing when nonantimonotone constraints are present is a great challenge because the aggregate functions do not increase or decrease monotonically along the subset relationship between partitions. In this paper, we propose a novel bound prune cubing (BP-Cubing) approach for iceberg cubing with nonantimonotone aggregation constraints. Given a cube over \( n \) dimensions, an aggregate for any group-by partition can be computed from the corresponding aggregation constraint over the most specific \( n \)-dimensional partitions (MSPs). The largest and smallest aggregate values computed this way become the bounds for all partitions in the cube. We provide efficient methods to compute tight bounds for base aggregate functions and, more interestingly, arithmetic expressions thereof, from bounds of aggregates over the MSPs. Our methods produce tighter bounds than those obtained by previous approaches. We present iceberg cubing algorithms that combine bounding with efficient aggregation strategies. Our experiments on real-world and artificial benchmark data sets demonstrate that BP-Cubing algorithms achieve more effective pruning and are several times faster than state-of-the-art iceberg cubing algorithms and that BP-Cubing achieves the best performance with the top-down cubing approach.

Index Terms—Data mining, data cube, pruning, data warehouses.

1 INTRODUCTION

With the multidimensional model for data warehouses, a data set consists of tuples over dimensions and measures, and queries involve aggregating the measures over partitions of tuples sharing identical dimension values. For example, the Structured Query Language (SQL) in Fig. 1a gives “the monthly total amount of sales for each city” on the Sales data set in Table 1. The (Month, City) group-by divides the data set into partitions of tuples sharing identical (Month, City) values, and the measure Sale is aggregated to yield \( \text{Sum}(\text{Sale}) \) for each partition. The Cube operator was proposed [6] to compute all potential group-bys of a data set, leading to the notion of a data cube.

The iceberg cube was proposed in [4], where only partitions whose aggregate values satisfy an aggregation constraint are produced. By adding “HAVING \( \text{Count}(*) \geq 10 \)” to the SQL query in Fig. 1a, we get an SQL query (in Fig. 1b) for computing those (Month, City) partitions, each having \( \geq 10 \) tuples. The aggregation constraint \( \text{Count}(*) \geq 10 \) is antimonotone. If a partition does not contain \( \geq 10 \) tuples and thus fails the constraint, then all its subpartitions will have fewer tuples and also fail the constraint. The bottom-up cubing (BUC) strategy was proposed [4], where aggregates are computed starting from the partition of all tuples and then followed recursively with the subpartitions, and antimonotone constraints are used for pruning.

In On-Line Analytical Processing (OLAP) applications, aggregation constraints often involve complex aggregates. The constraint \( \text{Sum}(x) \geq 5,000 \) and \( \text{Var}(x) \leq 100 \), which states that “the total profit \( x \) is at least $5,000, and the variance of profit is at most $100,” is a typical example of constraints for supporting enterprise data analysis and decision making. As \( x \) can be positive or negative, the subset relationship between a group and its subgroups may not imply monotonically increasing (or decreasing) aggregate values. Computing iceberg cubes with such constraints has been a challenging problem. Existing proposals in the literature have all followed the approach of deriving weaker but antimonotone constraints [7], [15]. They are restricted in two aspects: 1) the weaker constraints derived from nonantimonotone constraints are often very loose and not effective enough for pruning, especially for complex constraints, and 2) aggregates are computed following the BUC framework, and only one group is aggregated at a time in the recursive partitioning process, which limits the amount of information readily available for pruning.

In this paper, we propose a novel technique, called bound prune cubing (BP-Cubing), for efficiently computing iceberg cubes with nonantimonotone aggregation constraints. The main ideas are to effectively derive tight bounds for possible aggregates and to use such bounds to prune unproductive computation. An important part is played by the “most specific partitions” (MSPs), namely, those nonempty partitions that cannot be further divided (for the subcube under consideration). MSPs can be viewed as the

---

1. We use group and partition as synonyms.
SELECT Month, City, Sum(Sale) FROM Sales GROUP BY Month, City

SELECT Month, City, Sum(Sale) FROM Sales GROUP BY Month, City HAVING Count(*) \geq 10

(a) \hspace{1cm} (b)

Fig. 1. Two SQL group-by queries on the sales data set. (a) An SQL group-by. (b) Adding a constraint to (a).

Table 1

<table>
<thead>
<tr>
<th>Month</th>
<th>Product</th>
<th>Salesman</th>
<th>City</th>
<th>Sum(Sale)</th>
<th>Count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>Toy</td>
<td>John</td>
<td>Perth</td>
<td>200</td>
<td>5</td>
</tr>
<tr>
<td>March</td>
<td>TV</td>
<td>Peter</td>
<td>Perth</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>March</td>
<td>TV</td>
<td>John</td>
<td>Perth</td>
<td>100</td>
<td>20</td>
</tr>
<tr>
<td>March</td>
<td>TV</td>
<td>John</td>
<td>Sydney</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>April</td>
<td>TV</td>
<td>Peter</td>
<td>Perth</td>
<td>100</td>
<td>8</td>
</tr>
<tr>
<td>April</td>
<td>Toy</td>
<td>Peter</td>
<td>Sydney</td>
<td>100</td>
<td>5</td>
</tr>
</tbody>
</table>

Month, Product, Salesman, and City are dimensions. Sale is the measure.

in our experiments on real-world and synthetic benchmark data sets, our BP-Cubing algorithms are several times faster than existing pruning techniques, including the most recent Divide-and-Approximate (DnA) algorithm [15].

Organizationaly, Section 2 gives some preliminaries. Section 3 defines boundability of aggregate functions. We discuss how we can derive the bounds for base aggregates, their functions, and complex aggregates in Sections 4, 5, and 6, respectively. Section 7 then presents the BP-Cubing algorithms. We report experimental evaluations in Section 8. We discuss related works in Section 9 and conclude in Section 10.

2 SOME PRELIMINARIES ON DATA CUBES

We will use uppercase letters to denote dimensions and lowercase letters to denote dimension values. A group-by is a tuple of dimensions of the form \((A, B, C)\), and a partition of \((A, B, C)\) is a tuple of dimension values of the form \((a, b, c)\).

The group-bys in a data cube form a lattice structure called the cube lattice. Fig. 2 shows an example four-dimensional cube lattice. The empty group-by (which aggregates all tuples in a data set) is at the bottom of the lattice, and the group-by with all dimensions is at the top. The edges in the lattice represent subset-superset relationships between group-bys.

A special value “*” is used in specifying partitions, with the meaning that it can match any value (of the applicable dimension). For the data set in Table 1, \((\text{Jan}, *, *, *)\) denotes the partition of tuples having Month = January and having no restriction on the other dimensions. For brevity, “*” is usually omitted in naming partitions; for example, \((\text{Jan}, *, *, *)\) is written as \((\text{Jan})\).

The subset relationship also exists between partitions from different group-bys. For example, the 3D partition \((\text{Jan}, TV, Perth)\) is a subset of each of the 2D partitions \((\text{Jan}, TV)\), \((\text{Jan}, Perth)\), and \((TV, Perth)\).

Given a data set \(S\) of \(n\) dimensions \(A_1, \ldots, A_n\) and a measure \(X\), the corresponding data cube is usually referred to as Cube\((A_1, \ldots, A_n)\). Here, we think of the cube as the set of possible group-bys or the set of possible groups for all group-bys. For example, the four-dimensional cube in Table 1 can be denoted as Cube\((\text{Month}, \text{Product}, \text{Salesman}, \text{City})\).

For ease of discussion, we also refer to the cube as Cube\((X)\), and think of it as the “measure partitions” (the possible bags of measure values for the possible groups).
Specifically, let \( g_1, \ldots, g_m \) be all possible groups of the cube. By an abuse of notation, we will use \( X_i \) to denote the bag (multiset) \( \{ \{ X \} | l \in g_i \} \), and we will refer to \( X_1, \ldots, X_n \) as the measure partitions. Now, given an aggregate function \( F \), we apply \( F \) to the measure partitions in the data cube to get \( F(Cube(X)) = \{ F(X_i) | X_i \in Cube(X) \} \). With the aggregate function \( \text{Sum}(Sale) \), we can write \( \text{Sum}(Cube(Sale)) \) for the four-dimensional cube in Table 1.

Aggregate functions are categorized into distributive, algebraic, and holistic functions, depending on how an aggregate on a partition can be computed from aggregates on its subpartitions [6]. 1) An aggregate function \( F \) is distributive if there is a function \( G \) such that \( F(S) = G(\{ F(S_i) | i = 1, \ldots, n \} ) \) for a data set \( S \) and partitioning \( \{ S_1, \ldots, S_n \} \) of \( S \). For example, \( \text{Sum} \) and \( \text{Max} \) are distributive, with \( G = F \), and \( \text{Count} \) is distributive, with \( G = \text{Count} \). 2) An aggregate function \( F \) is algebraic if it can be computed by a function \( H \) with several arguments, each of which is obtained by applying a distributive aggregate function. \( \text{Avg} \), \( \text{Var} \), \( \text{Standard-Deviation} \), and \( \text{MaxN} \) and \( \text{MinN} \) are algebraic functions; for example, \( \text{Avg} \) is algebraic, since \( \text{Avg}(S) = \text{Sum}(S)/\text{Count}(S) \), and both \( \text{Sum} \) and \( \text{Count} \) are distributive. All distributive aggregates are algebraic. 3) An aggregate function \( F \) is holistic if it is not distributive or algebraic. \( \text{Rank} \), \( \text{Mode} \), and \( \text{Median} \) are examples of holistic functions. We will call the distributive aggregations used for computing algebraic functions \( F \) (item 2 above) the auxiliary aggregates.

Existing cubing algorithms have made use of the properties of distributive and algebraic functions to compute supergroups from subgroups [1], [10], [11], [17]. No efficient cubing algorithms for holistic functions have been reported. It is worth noting that in most applications, aggregate cubing algorithms for holistic functions have been reported. Super-aggregates from subgroups [1], [10], [11], [17]. No efficient cubing algorithms for holistic functions have been reported. The tightest upper bound of 700 and the tightest lower bound are, respectively, reached by the largest and smallest aggregate values that can be produced by any set of MSPs of the data cube.

Definition 2 (Aggregate function bound). An upper bound of an aggregate function \( F \) for the data cube on \( X \) is a real number \( U \) such that for any partition \( X_i \) of \( Cube(X) \), it is the case that \( F(X_i) \leq U \). Similarly, we can define a lower bound for \( F(Cube(X)) \) to be a real number \( L \) such that for any partition \( X_i \), \( F(X_i) \geq L \).

Observation 1. Given a data cube on measure \( X \) and an aggregate function \( F \), the tightest upper bound and lower bound are, respectively, reached by the largest and smallest aggregate values that can be produced by any set of MSPs of the data cube.

Example 1. In Table 1, \( \text{Sum}(Sale) \) for any group in the data cube is not larger than the sum of \( \text{Sum}(Sale) \) for all six MSPs, which is 700. On the other hand, \( \text{Sum}(Sale) \) for any group in the data cube is not smaller than the minimal \( \text{Sum}(Sale) \) among the six MSPs, which is 100. Therefore, \( \text{Sum}(Sale) \) has the tightest upper bound of 700 and the tightest lower bound of 100.

The notion of MSP and data cube core is also applied to subdata cubes. Fig. 2a shows \( Cube(ABCD) \), and its MSPs are the ABCD partitions. The polygon on the left of Fig. 2a denotes a set of lattice structures, as shown in Fig. 2b. The polygon on the right of Fig. 2a represents \( Cube(BCD) \). The MSPs for \( Cube(BCD) \) are BCD partitions, which are unions of ABCD partitions, with equal values for B, C, and D. The lattice structure, as shown in Fig. 2b, is called a subdata cube. The lattice structure in Fig. 2b, \( (a_1, B, C, D) \) partitions, which are a subset of the ABCD partitions, form its core. The relationship between cores for a cube and its subcubes, and that for a cube and its lower-dimensional counterparts, are important for computing bounds from MSPs, without incurring extra cost and for effective pruning with bounds, as shown in our BP-Cubing algorithms, discussed in Section 7. In later discussions, we use the term “data cube” to refer to a standard or subdata cube.

Definition 3 (Subdata cube). Consider dimensions \( A_1, \ldots, A_n \) and \( B_1, \ldots, B_k \) and values

\[ b_1 \in \text{domain}(B_1), \ldots, b_k \in \text{domain}(B_k) \]

The groups aggregating tuples that satisfy \( B_i = b_i \) for \( 1 \leq i < k \) form a subdata cube, or simply a subcube, denoted as \( Cube(A_1, \ldots, A_n | b_1, \ldots, b_k) \). The core of the subdata cube comprises MSPs with \( B_i = b_i \) for \( 1 \leq i < k \). \( B_1, \ldots, B_k \) are conditional dimensions for the subcube.

Example 2. Continuing with Example 1, consider

\[ Cube(Product, SalesMan, City) | Mar. \]

There are three MSPs for the data cube: (Mar, TV, Peter, Perth), (Mar, TV, John, Perth), and
tighter bounds can be obtained on subcubes. Indeed, the tightest upper bound for \( \text{Sum}(\text{Sale}) \) for the data cube is \( 100 + 100 + 100 = 300 \), and the tightest lower bound is \( \text{Min}\{100, 100, 100\} = 100 \).

Based on Observation 1, we can see that a data cube can be bounded from its MSPs. However, exhaustively checking the power set of MSPs is equivalent to computing the complete data cube and is not computationally feasible. We define boundable aggregate functions as follows.

**Definition 4 (Boundable aggregate function).** An aggregate function \( F \) is boundable for a data cube if some upper and lower bounds of \( F \) for the data cube can be determined by some algorithm with a single scan of some auxiliary aggregate values of the MSPs of the data cube. We will use \( F'(\text{Cube}(X)) \) and \( F(\text{Cube}(X)) \) to denote, respectively, the upper and lower bounds computed by a given single-scan algorithm.

It should be pointed out that the bounds computed may not be the tightest. To ensure the effectiveness of pruning, we aim to derive bounds that are as tight as possible. Let us use some example aggregate functions to explain this definition.

**Example 3.** Consider the aggregate function Count and measure \( X \) with \( n \) MSPs \( X_1, \ldots, X_n \), where

\[
\text{Count}(X) = \text{Sum}\{\text{Count}(X_i) | i = 1 \ldots n\}.
\]

The auxiliary aggregate function is Count. For any partition \( g \) of \( \text{Cube}(X) \), the number of tuples in \( g \) is not larger than the total number of tuples of all MSPs; in other words, \( \text{Count}(g) \leq \text{Sum}\{\text{Count}(X_i) | i = 1 \ldots n\} \).

On the other hand, we also have

\[
\text{Count}(g) \geq \text{Min}\{\text{Count}(X_i) | i = 1 \ldots n\}.
\]

As a result, \( \text{Sum}\{\text{Count}(X_i) | i = 1 \ldots n\} \) is an upper bound, and \( \text{Min}\{\text{Count}(X_i) | i = 1 \ldots n\} \) is a lower bound. They can be obtained by a single scan of the auxiliary aggregates of MSPs. Therefore, \( \text{Count}(\text{Cube}(X)) \) is boundable.

We now give an example of aggregate functions whose bounds can be infinite.

**Example 4.** Given measure \( X \) and aggregate function \( 1/\text{Sum} \), consider \( \text{Cube}(X) \) with MSPs \( X_1, \ldots, X_n \). Then, \( 1/\text{Sum}(X) = 1/\text{Sum}\{\text{Sum}(X_i) | i = 1 \ldots n\} \). As will be explained later in Tables 3 and 4, the signs of \( \text{Sum}(X_i) \) are important for computing the bounds:

1. If \( \text{Sum}(X_i) > 0 \) for all \( i = 1 \ldots n \), then the upper and lower bounds are computed from, respectively, the positive lower and upper bounds of \( \text{Sum}(\text{Cube}(X)) \):

\[
1/\text{Sum}(\text{Cube}(X)) = 1/\text{Min}\{\text{Sum}(X_i) | i = 1 \ldots n\}
\]

2. If \( \text{Sum}(X_i) < 0 \) for all \( i = 1 \ldots n \), then the upper and lower bounds are computed from, respectively, the negative lower and upper bounds of \( \text{Sum}(\text{Cube}(X)) \):

\[
1/\text{Sum}(\text{Cube}(X)) = 1/\text{Max}\{\text{Sum}(X_i) | i = 1 \ldots n\}
\]

3. For general cases, where \( \text{Sum}(X_i) \) can be positive or negative, the upper bound \( 1/\text{Sum}(\text{Cube}(X)) \) is computed from the smallest \( \text{Sum}(x) > 0 \) for any partition in the cube. For any group \( g \) of \( \text{Cube}(X) \), \( \text{Sum}(g) \) can be the sum of any positive or negative MSP aggregates and can have a minimum of 0. Thus, the upper bound for \( 1/\text{Sum}(\text{Cube}(X)) \) is \( \infty \). Similarly, the lower bound \( 1/\text{Sum}(\text{Cube}(X)) \) is computed from the largest \( \text{Sum}(x) < 0 \) for any partition in the cube, which can be a negative approaching 0. Therefore, the lower bound for \( 1/\text{Sum}(\text{Cube}(X)) \) is \( -\infty \).

Computing the upper and lower bounds for arithmetic expressions of base aggregate functions such as \( 1/\text{Sum} \) is summarized later in Table 4.

### Table 2: The Bounds of Base Aggregate Functions

<table>
<thead>
<tr>
<th>( F )</th>
<th>( F'(\text{Cube}(X)); F(\text{Cube}(X)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>( \text{Sum}(\text{Cube}(X)); \text{Min Count}(X) )</td>
</tr>
<tr>
<td>Max</td>
<td>( \text{Max Max}(X_i); \text{Min Max}(X_i) )</td>
</tr>
<tr>
<td>Min</td>
<td>( \text{Max Min}(X_i); \text{Min Min}(X_i) )</td>
</tr>
<tr>
<td>Sum</td>
<td>( \text{Sum}(\text{Cube}(X)); \text{Sum}(X_i) ) if ( \text{Sum}(X_i) &gt; 0 ); ( \text{Max Sum}(X_i) ) otherwise |</td>
</tr>
<tr>
<td></td>
<td>( \text{Min Sum}(X_i) ) otherwise</td>
</tr>
</tbody>
</table>

Given a data set, \( X \) is the measure, and \( X_1, \ldots, X_n \) are the MSPs. (a) There is \( i \) such that \( \text{Sum}(X_i) > 0 \). (b) There is \( i \) such that \( \text{Sum}(X_i) < 0 \).

\[
1/\text{Sum}(\text{Cube}(X)) = 1/\text{Sum}\{\text{Sum}(X_i) | i = 1 \ldots n\}.
\]

4 **Bounding Base Aggregate Functions**

We now consider how we can bound the base aggregate functions of SQL.

**Theorem 1.** The base aggregate functions Count, Min, Max, and Sum are boundable, and their bounds are listed in Table 2.

We use the following example for Sum to explain the reasoning for the bounds of Table 2 and how they are obtained by a single scan of MSPs.

**Example 5.** Suppose we are to compute a data cube on measure \( X \), which may take positive or negative values. Let the MSPs of \( X \) be \( X_1, \ldots, X_n \). With a single scan of \( X_1, \ldots, X_n \), we have \( \text{Sum}(X_i)(1 \leq i \leq n) \) and the positive/negative and maximal/minimal values among them:
The Sign Bounds of Base Aggregate Functions

<table>
<thead>
<tr>
<th>$F$</th>
<th>$F^+(\text{Cube}(X))$; $F^+(\text{Cube}(X))$</th>
<th>$F^-(\text{Cube}(X))$; $F^-(\text{Cube}(X))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>$\text{Sum}(\text{Count}(X) \cup \text{Min}(\text{Count}(X)))$</td>
<td>$n/a ; n/a$</td>
</tr>
<tr>
<td>Max</td>
<td>$\max(\max(X_i); \min(X_i))$</td>
<td>$\max(X_i)$; $\min(X_i)$</td>
</tr>
<tr>
<td>Min</td>
<td>$\min(\max(X_i); \min(X_i))$</td>
<td>$\max(X_i); \min(X_i)$</td>
</tr>
<tr>
<td>Sum</td>
<td>$\text{Sum}(\text{Sum}(X_i); \text{Sum}(X_i) &gt; 0)$</td>
<td>$\text{Max}(\text{Sum}(X_i); \text{Sum}(X_i) &gt; 0)$; $\text{Min}(\text{Sum}(X_i); \text{Sum}(X_i) &gt; 0)$</td>
</tr>
</tbody>
</table>

$X$ is the measure of a data set, and the MSPs for the data cube are $X_1, \ldots, X_n$.

1. If there exists $i$ such that $\text{Sum}(X_i) > 0$, then the sum for any group in $\text{Cube}(X)$ is not greater than the sum of all such positive $\text{Sum}(X_i)$. Otherwise, $\text{Sum}(X_i) \leq 0$ for $i = 1 \ldots n$; the sum for any group is not greater than the maximal $\text{Sum}(X_i)$. Thus,

$$\text{Sum}(\text{Cube}(X)) = \max_{i} \text{Sum}(X_i)$$

if there exists $\text{Sum}(X_i) > 0$;

$$\text{Sum}(\text{Cube}(X)) = \text{Max}_{i} \text{Sum}(X_i), \text{ otherwise.}$$

2. If there exists $i$ such that $\text{Sum}(X_i) < 0$, then the sum of any group in $\text{Cube}(X)$ is not less than the sum of all such $\text{Sum}(X_i)$. Otherwise, $\text{Sum}(X_i) \geq 0$ for all $i = 1 \ldots n$; the sum of any group is not less than the minimal $\text{Sum}(X_i)$. Therefore,

$$\text{Sum}(\text{Cube}(X)) = \min_{i} \text{Sum}(X_i),$$

if there exists $\text{Sum}(X_i) < 0$;

$$\text{Sum}(\text{Cube}(X)) = \text{Min}_{i} \text{Sum}(X_i), \text{ otherwise.}$$

To bound functions that are arithmetic expressions of base aggregate functions, sometimes, the sign bounds of aggregations need to be computed (see Section 5). These include positive upper bound ($F^+$), positive lower bound ($F^-$), negative upper bound ($F^-$), and negative lower bound ($F^-$), where $F$ is either a base aggregate function Min, Max, Sum, or Count or an arithmetic expression of base aggregate functions. $F^+(\text{Cube}(X))$ is defined to be a real $l \geq 0$ obtained by a single-scan algorithm such that $l \geq F(g) \geq 0$ for any partition $g$ in $\text{Cube}(X)$. The other sign bounds can be defined similarly. The positive (negative) bounds are applicable only if there are nonnegative (negative) aggregates among the MSPs. The sign bounds of base aggregate functions are listed in Table 3.

We explain how the sign bounds for Sum are computed. Computation for other functions is straightforward. We discuss how we can compute the positive bounds of Sum, with discussions on the negative bounds mirror the same case as in positive bounds. The positive upper bound is the sum of all positive sums of the MSPs. To compute the positive lower bound, two cases exist: 1) if $\text{Sum}(X_i) > 0$ for all $X_i (1 \leq i \leq n)$, then the positive lower bound is the minimal $\text{Sum}(X_i) \geq 0$ and 2) if the sums of MSPs include both negative and positive ones, then the sum of any set of MSPs can be as low as 0, and so, 0 is the positive lower bound.

5 Bounding Arithmetic Expressions of Base Aggregate Functions

Consider a data cube $\text{Cube}(X)$ and an arithmetic expression of auxiliary aggregates $E = E_1 \circ E_2$, where $E_1$ and $E_2$ are base aggregate functions or arithmetic expressions of base aggregates. If the operator is “$+$” or “$-$”, then the upper (lower) bounds of $E(\text{Cube}(X))$ can be computed from the upper (lower) bounds of $E_1(\text{Cube}(X))$ and $E_2(\text{Cube}(X))$. If the operator is “$\times$” or “$/$”, then we need to use the sign bounds of $E_1(\text{Cube}(X))$ and $E_2(\text{Cube}(X))$ to bound $E(\text{Cube}(X))$.

Proposition 1. Given $\text{Cube}(X)$ on measure $X$ and an arithmetic expression $E = E_1 \circ E_2$, where the operator is “$+$”, “$-$”, “$\times$”, or “$/$”, and $E_1$ and $E_2$ are base aggregate functions or arithmetic expressions of base aggregates, $E(\text{Cube}(X))$ can be computed from the bounds or sign bounds of $E_1(\text{Cube}(X))$ and $E_2(\text{Cube}(X))$, as shown in Table 4.

Proof. Let $X_1, \ldots, X_n$ be the MSPs of $\text{Cube}(X)$. We consider each of the four possible operators:

1. $E_1 + E_2$. By the definition of upper bound, for any group $g \in \text{Cube}(X)$, $E_1(g) \leq E_1(\text{Cube}(X))$, and $E_2(g) \leq E_2(\text{Cube}(X))$. We have

$$E_1(g) + E_2(g) \leq E_1(\text{Cube}(X)) + E_2(\text{Cube}(X)).$$

Thus,

$$(E_1 + E_2)(\text{Cube}(X)) = E_1(\text{Cube}(X)) + E_2(\text{Cube}(X)).$$

The proof is similar for

$$(E_1 + E_2)(\text{Cube}(X)) = E_1(\text{Cube}(X)) + E_2(\text{Cube}(X)).$$
2. \(E_1 - E_2\). Given \(g \in \text{Cube}(X)\), \(E_1(g) \leq E_1'(\text{Cube}(X))\) and \(E_2(g) \geq E_2'(\text{Cube}(X))\). Thus,

\[
E_1(g) - E_2(g) \leq E_1'(\text{Cube}(X)) - E_2'(\text{Cube}(X))
\]

and

\[
(E_1 - E_2)\text{Cube}(X) = E_1'(\text{Cube}(X)) - E_2'(\text{Cube}(X)).
\]

The proof is similar for

\[
(E_1 - E_2)\text{Cube}(X) = E_1'(\text{Cube}(X)) - E_2'(\text{Cube}(X)).
\]

3. \(E_1 \times E_2\). Let \(g\) be a group of \(\text{Cube}(X)\). Let \(Y_1, Y_2, \ldots, Y_k\) be the MSPs that are contained in \(g\). Based on their signs under \(E_1\), we form two subsets from these MSPs:

\[
P \mathcal{E}_1 = \{Y_j | E_1(Y_j) \geq 0, 1 \leq j \leq k\},
\]

\[
N \mathcal{E}_1 = \{Y_j | E_1(Y_j) < 0, 1 \leq j \leq k\}.
\]

Similarly, based on their signs under \(E_2\), we form the following subsets from these MSPs:

\[
P \mathcal{E}_2 = \{Y_j | E_2(Y_j) \geq 0, 1 \leq j \leq k\},
\]

\[
N \mathcal{E}_2 = \{Y_j | E_2(Y_j) < 0, 1 \leq j \leq k\}.
\]

For the upper bound, we first consider the case where \(P \mathcal{E}_1 \neq \emptyset\) and \(P \mathcal{E}_2 \neq \emptyset\) or \(N \mathcal{E}_1 \neq \emptyset\) and \(N \mathcal{E}_2 \neq \emptyset\). \(E_1(g) \times E_2(g)\) can have positive values. Thus,

\[
E_1(g) \times E_2(g) \leq \max \left(\frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right).
\]

Otherwise, \(E_1(g) \times E_2(g)\) can only be negative. We have

\[
E_1(g) \times E_2(g) \leq \min \left(\frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right).
\]

For all cases, by combining the two inequalities with \(E_1(g) \times E_2(g)\) on the left-hand side, we get \((E_1 \times E_2)\text{Cube}(X)\):

\[
\max \left(\frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right), \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right).
\]

The proof for the lower bound is quite similar to that for the upper bound. The possible positive values of \(g\) are

\[
\geq \min \left(\frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right), \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right).
\]

and the possible negative values of \(g\) are

\[
\geq \min \left(\frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right), \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right).
\]

By combining the last two inequalities, we get

\[
(E_1 \times E_2)\text{Cube}(X)\):
\]

\[
\min \left(\frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right), \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}, \frac{E_1'(\text{Cube}(X))}{E_1'(\text{Cube}(X))} \times \frac{E_2'(\text{Cube}(X))}{E_2'(\text{Cube}(X))}\right).
\]

4. \(E_1 / E_2\). The proof is similar to that for the operator "," and is omitted. □

Example 6. Consider a data cube \(\text{Cube}(X)\), where the measure \(X\) can be positive or negative, and the aggregate function \(\text{Avg}\). Suppose the MSPs are \(X_1, \ldots, X_n\). For any MSP \(X_i\), we note that \(\text{Avg}(X_i) = \text{Sum}(X_i)/\text{Count}(X_i)\), and \(\text{Count}(X_i)\) is always positive. Following Table 4, \(\text{Avg}(\text{Cube}(X))\) is

\[
\text{Max}\left(\frac{\text{Sum}^+\left(\text{Cube}(X)\right)}{\text{Count}^+\left(\text{Cube}(X)\right)}\right), \frac{\text{Sum}^-\left(\text{Cube}(X)\right)}{\text{Count}^+\left(\text{Cube}(X)\right)}\right).
\]

and \(\text{Avg}(\text{Cube}(X))\) is

\[
\text{Min}\left(\frac{\text{Sum}^+\left(\text{Cube}(X)\right)}{\text{Count}^+\left(\text{Cube}(X)\right)}\right), \frac{\text{Sum}^-\left(\text{Cube}(X)\right)}{\text{Count}^+\left(\text{Cube}(X)\right)}\right).
\]

If there exists some \(i\) such that \(\text{Sum}(X_i) > 0\), then

\[
\text{Avg}(\text{Cube}(X)) = \frac{\text{Sum}^+\left(\text{Cube}(X)\right)}{\text{Count}(\text{Cube}(X))}, \text{Avg}(\text{Cube}(X)) = \frac{\text{Sum}^-\left(\text{Cube}(X)\right)}{\text{Count}(\text{Cube}(X))}.
\]

Otherwise, \(\text{Sum}(X_i) \leq 0\) for all \(i = 1 \ldots n\); then, we have

\[
\text{Avg}(\text{Cube}(X)) = \frac{\text{Sum}^+\left(\text{Cube}(X)\right)}{\text{Count}(\text{Cube}(X))}, \text{Avg}(\text{Cube}(X)) = \frac{\text{Sum}^-\left(\text{Cube}(X)\right)}{\text{Count}(\text{Cube}(X))}.
\]

We then follow Table 3 to compute the sign bounds in the above expressions.

The sign bounds for base aggregate functions are listed in Table 3. We now discuss how we can derive the sign bounds for arithmetic expressions. Note that in Table 5, positive (negative) bounds are applicable only if there exists an MSP \(X_i\) in the cube such that

\[
(E_1(X_i) \text{ op } E_2(X_i)) \geq 0 \text{ or } (E_1(X_i) \text{ op } E_2(X_i)) < 0.
\]
Table 5: The Sign Bounds of Arithmetic Expressions

<table>
<thead>
<tr>
<th>op</th>
<th>(E1 op E2)⁺; (E1 op E2)⁻</th>
<th>(E1 op E2)⁺; (E1 op E2)⁻</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>E1 + E2; if E1 + E2 ≥ 0, E1 + E2, otherwise 0.</td>
<td>E1 + E2; if E1 + E2 &lt; 0, E1 + E2, otherwise 0.</td>
</tr>
<tr>
<td>−</td>
<td>E1 − E2; if E1 − E2 ≥ 0, E1 − E2, otherwise 0.</td>
<td>E1 − E2; if E1 − E2 &lt; 0, E1 − E2, otherwise 0.</td>
</tr>
<tr>
<td>×</td>
<td>Max(E1⁺×E2⁺; E1⁺×E2⁻; E1⁻×E2⁺); Min(E1⁺×E2⁺; E1⁺×E2⁻; E1⁻×E2⁺)</td>
<td>Max(E1⁺×E2⁺; E1⁺×E2⁻; E1⁻×E2⁺); Min(E1⁺×E2⁺; E1⁺×E2⁻; E1⁻×E2⁺)</td>
</tr>
<tr>
<td>÷</td>
<td>Max(E1⁺/E2⁺; E1⁺/E2⁻; E1⁻/E2⁺); Min(E1⁺/E2⁺; E1⁺/E2⁻; E1⁻/E2⁺)</td>
<td>Max(E1⁺/E2⁺; E1⁺/E2⁻; E1⁻/E2⁺); Min(E1⁺/E2⁺; E1⁺/E2⁻; E1⁻/E2⁺)</td>
</tr>
</tbody>
</table>

Short notations E₁, E₂, and (E₁ op E₂) denote E₁(Cube(X)), E₂(Cube(X)), and (E₁ op E₂)(Cube(X)), respectively, where X is the measure of a data set.

For example, the positive upper bound

[(E₁ + E₂)(Cube(X))⁺]

is applicable only if there exists X; such that

E₁(X) + E₂(X) ≥ 0.

For the operators “+” and “−”, when E₁ + E₂ or E₁ − E₂ can be positive or negative for MSPs, the positive lower and negative upper bounds for both expressions are 0. This is because the combination of positive and negative MSPs cannot produce the aggregate value of 0. For the operators “×” and “÷”, the signs of E₁ and E₂ for MSPs decide the applicable computation. We use an example to explain Table 5.

Example 7. Given data cube Cube(X) with MSPs X₁, ..., Xₙ, consider the aggregate function

E(X) = 1/(Max(X) + Min(X)).

To compute E(Cube(X)), three cases can arise:

Case 1. Min(Xᵢ) ≥ 0 for all i = 1, ..., n. It follows that Max(Xᵢ) ≥ 0 for all i = 1, ..., n. Then,

E(Cube(X)) = 1/Max⁺(Cube(X)) + Min⁺(Cube(X)).

Case 2. Max(Xᵢ) ≤ 0 and Min(Xᵢ) ≤ 0 for all i = 1, ..., n. Then,

E(Cube(X)) = 1/Max⁻(Cube(X)) + Min⁻(Cube(X)).

Case 3. Max(Xᵢ) and Min(Xᵢ) can be positive or negative (1 ≤ i ≤ n). Based on Table 4, we have E(Cube(X)) = 1/(Max(X) + Min(X))⁺(Cube(X)). Based on Table 5, (Max(X) + Min(X))⁻(Cube(X)) is 0, which means that the upper bound for 1/(Max(X) + Min(X))(Cube(X)) is ∞.

Proposition 1 leads to the following theorem about bounding arithmetic expressions of base aggregate functions.

Theorem 2. Given a data cube on measure X, an arithmetic expression E = E₁ op E₂ of base aggregate functions, where the operator is +, −, ×, or /, E is boundable for Cube(X), and its bounds can be computed following Tables 2, 3, 4, and 5.

Proof. By Proposition 1, we can compute the bounds for an arithmetic expression from the bounds or sign bounds of the operand expressions, as listed in Table 4. Table 5 shows how the sign bounds of any expression can be computed from the bounds for base aggregate functions. Tables 2 and 3 demonstrate that a single scan of the MSPs of a data cube can compute the bounds of all base aggregate functions. As a result, all arithmetic expressions of base aggregate functions are boundable, and the bounds can be computed following Tables 2, 3, 4, and 5.

We now show how we can compute the upper bound for the complex aggregate function Var based on the theorem.

Example 8. The Var for X = {xᵢ | i = 1, ..., N} is defined as

Var(X) = Σ(i=1 to N) xᵢ² / N,

where X denotes the Avg of X. It can be rewritten as follows:

Var(X) = Σ(i=1 to N) xᵢ² / N - Σ(i=1 to N) xᵢ / N × Σ(i=1 to N) xᵢ / N.

Let QSum(X) denote Σ(i=1 to N) xᵢ². It follows that Var(X) is an arithmetic expression of QSum(X), Sum(X), and Count(X). Consider a data cube Cube(X) with MSPs X₁, ..., Xₙ. Based on Table 4, we have

Var(Cube(X)) = Σ(i=1 to N) xᵢ² / Count(X) - Σ(i=1 to N) xᵢ / Count(X)².

We next consider each of the two subexpressions separately.

We first consider Σ(i=1 to N) xᵢ² / Count(X): For any MSP Xᵢ, QSum(Xᵢ) and Count(X) are always positive. By following Table 4, we have the following fractional expression, where the numerator and denominator expressions can be computed following Table 2:

QSum(X) / Count(X) = QSum(Cube(X)) / Count(Cube(X)).

We then consider Σ(i=1 to N) xᵢ / Count(X)²: Based on Table 4, we have

(Σ(i=1 to N) xᵢ / Count(X)²) ≥ Min((Σ(i=1 to N) xᵢ / Count(X))⁺ × (Σ(i=1 to N) xᵢ / Count(X))⁻).

Count is always positive. By following Table 5 for computing sign bounds, we get (Σ(i=1 to N) xᵢ / Count(X))² (Cube(X)).

8. There is another definition of Variance: Var(X) = Σ(i=1 to N) xᵢ² / N. The discussions here also apply to this definition.
Min\left(\frac{\text{Sum}^+(\text{Cube}(X))}{\text{Count}(\text{Cube}(X))}\right)^2, \quad \left(\frac{\text{Sum}^-(\text{Cube}(X))}{\text{Count}(\text{Cube}(X))}\right)^2.

In the above equation,
\text{Count}(\text{Cube}(X)) = \sum_i \text{Count}(X_i).

Moreover, based on Table 3,
\text{Sum}^+(\text{Cube}(X)) = \min \sum_i \text{Sum}(X_i)

if \text{Sum}(X_i) \geq 0 for all \(i = 1 \ldots n\); otherwise, it is 0.
Similarly, \text{Sum}^-(\text{Cube}(X)) = \max \sum_i \text{Sum}(X_i) if \text{Sum}(X_i) < 0 for all \(i = 1 \ldots n\); otherwise, it is 0.

In summary, we compute \text{Var}(\text{Cube}(X)) as follows:

if \text{Sum}(X_i) \geq 0 for \(i = 1 \ldots n\),
\text{Sum} \sum_i \frac{\text{Sum}(X_i^2)}{\text{Count}(X_i)} - \left(\frac{\text{Min} \sum_i \text{Sum}(X_i)}{\text{Sum} \text{Count}(X_i)} \right)^2; 

otherwise, if \text{Sum}(X_i) < 0 for \(i = 1 \ldots n\),
\text{Sum} \sum_i \frac{\text{Sum}(X_i^2)}{\text{Count}(X_i)} - \left(\frac{\text{Max} \sum_i \text{Sum}(X_i)}{\text{Sum} \text{Count}(X_i)} \right)^2;

otherwise,
\text{Sum} \sum_i \frac{\text{Sum}(X_i^2)}{\text{Count}(X_i)}.

With these equations, obviously tighter \text{Var}(\text{Cube}(X)) is obtained if the sums of MSPs are of the same sign.

6 Optimizations for Bounding Complex Functions

When rewriting several commonly used algebraic aggregation functions, namely, \text{Avg}, \text{Var}, and \text{Standard-Deviation}, into arithmetic expressions of distributive aggregate functions, we notice that very often, they use the following subexpression:

\(F(X) = \sum_i G_1(X_i)/\sum_i G_2(X_i)\),

where \(G_1\) and \(G_2\) are some distributive aggregate functions, and \(X_1, \ldots, X_n\) are MSPs. We can rewrite \text{Avg}(X) as \(\text{Sum}(X)/\text{Count}(X)\) and \text{Var}(X) as

\(\text{QSum}(X)/\text{Count}(X) - (\text{Sum}(X)/\text{Count}(X))^2\).

Moreover, we have
\(\text{Sum}(X)/\text{Count}(X) = \sum_i \text{Sum}(X_i)/\sum_i \text{Count}(X_i)\), which is \(F\), with \(G_1 = \text{Sum}\) and \(G_2 = \text{Count}\). Therefore, it is desirable to give tighter bounds for \(F\) than those obtained by Table 4. The next result provides not only tight bounds but also the optimal.

Theorem 3. Let \text{Cube}(X) be a data cube over measure \(X\) with MSPs \(X_1, \ldots, X_n\), let \(G_1\) and \(G_2\) be aggregate functions, and let \(F\) be as defined above. If \(G_2(X_i) > 0\) for all \(i = 1 \ldots n\), then we can bound \(F\) using

\(F'(\text{Cube}(X)) = \max_i F(X_i), F'(\text{Cube}(X)) = \min_i F(X_i)\).

Moreover, these bounds are the optimal bounds for \(F'(\text{Cube}(X))\).

Proof. Without loss of generality, we can assume that \(\min F(X_i) = F(X_1)\), and \(\max F(X_i) = F(X_n)\). Given a group \(g\) of \text{Cube}(X), let \(X_{g_1}, \ldots, X_{g_k}\) be the MSPs contained in \(g\). Thus, \(g = X_{g_1} \cup \ldots \cup X_{g_k}\).

\(F(g) = \sum_i G_1(X_{g_i})/\sum_i G_2(X_{g_i}), = \frac{G_1(X_{g_1}) + \ldots + G_1(X_{g_k})}{G_2(X_{g_1}) + \ldots + G_2(X_{g_k})}, = \frac{F(X_{g_1}) \times G_2(X_{g_1}) + \ldots + F(X_{g_k}) \times G_2(X_{g_k})}{G_2(X_{g_1}) + \ldots + G_2(X_{g_k})}.

The division in the last step is permitted, since \(G_2(X_{g_i}) > 0\) for all \(i\). Therefore,

\(F(g) - F(X_n) = \frac{F(X_{g_1}) \times G_2(X_{g_1}) + \ldots + F(X_{g_k}) \times G_2(X_{g_k})}{G_2(X_{g_1}) + \ldots + G_2(X_{g_k})} - F(X_n) \times G_2(X_n) + \ldots + G_2(X_n) = \frac{F(X_{g_1}) - F(X_n)}{G_2(X_{g_1}) - G_2(X_n)} + \ldots + \frac{F(X_{g_k}) - F(X_n)}{G_2(X_{g_k}) - G_2(X_n)}.

Since \(F(X_{g_j}) \leq F(X_n)\) and \(G_2(X_{g_j}) > 0\) for all \(j = 1 \ldots k\), we have \(G_2(X_{g_1}) + \ldots + G_2(X_{g_k}) > 0\) and \((F(X_n) - F(X_{g_j})) \times G_2(X_{g_j}) \leq 0\) for all \(j\). Therefore, \(F(g) - F(X_n) \leq 0\) or, equivalently, \(F(g) \leq F(X_n)\). Similarly, it can be shown that \(F(g) \geq F(X_1)\). Thus, we have proven that \(F(X_n)\) and \(F(X_1)\) are the upper and lower bounds for \(F'(\text{Cube}(X))\), respectively.

As \(\min F(X_i)\) is the smallest aggregate for an MSP and thus is a real aggregate in \(F'(\text{Cube}(X))\), all other lower bounds for \(F'(\text{Cube}(X))\) must be greater than \(\min F(X_i)\). \(\min F(X_i)\) is thus the tightest lower bound for \(F'(\text{Cube}(X))\). Similarly, we can prove that \(\max F(X_i)\) is the tightest upper bound for \(F'(\text{Cube}(X))\).

Next, we show that the bounds obtained by following Tables 2 and 4 are weaker. Considering \(\max F(X_i)\) (and still assuming that \(\max F(X_i) = F(X_n)\)), the case for \(\min F(X_i)\) is similar. By following Tables 2 and 4, \(F'(\text{Cube}(X))\) is computed:

**Case 1.** There exists \(j\) such that \(G_1(X_j) > 0\).

Thus, \(F'(\text{Cube}(X))\) is

\(\sum_i G_1(X_i)/\sum_i G_2(X_i) = \sum_i G_1(X_i)/\min_i G_2(X_i)\).

Since \(G_2(X_i) > 0\) for all \(i = 1 \ldots n\), and \(G_1(X_j) > 0\), \(F'(X_j) > 0\). Since \(\max F(X_i) = F(X_n)\), it follows that \(G_1(X_n) > 0\). Since \(\sum_i G_1(X_i) \geq G_1(X_n)\) and \(\min_i G_2(X_i) \leq G_2(X_n)\), we have

\(\sum_i G_1(X_i)/\min_i G_2(X_i) \geq \frac{G_1(X_n)}{G_2(X_n)} = F(X_n)\).
The auxiliary aggregates necessary to compute the iceberg cube. In our example, the auxiliary aggregates in a node are \( \text{Sum}(\text{Sale}) \) and \( \text{Count}(\ast ) \). For the leftmost path from the root of the G-tree in Fig. 3, the node \( \text{March} \) shows that there are 70 tuples with \( \text{Sum}(\text{Sale}) = 300 \) in the \( \text{(March,*,*,*)} \) partition, whereas the node \( \text{Peter} \) shows that there are 40 tuples with \( \text{Sum}(\text{Sale}) = 100 \) in the \( \text{(March,TV,Peter,*)} \) partition.

For a given data set, different dimension orders result in different G-trees. Depending on whether the G-tree is traversed top-down or bottom-up in computing cubes, the tree should be constructed in different orders of dimensions. Antimonotone pruning is effective when the most discriminating dimension is examined first. This suggests the cardinality-descending order of dimensions during cube computation. As a result, for bottom-up traversal, the cardinality-ascending order should be used in constructing the G-tree. In contrast, for top-down traversal, the cardinality-descending order should be used in constructing the G-tree. Our experiments have confirmed that such heuristics indeed have a positive impact on the effectiveness of pruning and the efficiency of cubing algorithms.

### 7 Bound-Prune Cubing Algorithms

We first present the group tree (G-tree) data structure that we will use for iceberg cubing. We then explain how bound prune cubing (BP-Cubing) is implemented on the G-tree. Finally, we briefly discuss how our algorithms utilize antimonotone constraints. The iceberg cube with the constraint “\( \text{Avg}(\text{Sale}) \) in [15, 20]” on the Sales data set in Table 1 is used as a running example throughout this section.

#### 7.1 The G-Tree

The underlying data structure for BP-Cubing is the G-tree. A G-tree is the compression of a given input data set and is constructed by one scan of the data set. It is used for both the top-down and bottom-up bound prune cubing algorithms (BP-Cubing(TD) and BP-Cubing(BU), respectively), although there are some minor differences depending on the traversal strategy.

The G-tree for the data set in Table 1 is shown in Fig. 3. A G-tree for an \( n \)-dimensional data set is of depth \( n \), where each level represents a dimension. A path starting from the root collapses the tuples with common dimension values along the path. Each tree node keeps...
Consider the G-tree with top-down aggregation, given a G-tree in the observation below. compute the bounds and use them for pruning, as described in the order in which trees are built.

G-trees are constructed to compute all group-bys in a data cube. When constructing a G-tree for an n-dimensional data set, we simultaneously compute n group-bys, namely, the group-bys whose dimensions are prefixes of the list of dimensions ordered by the levels of the tree. To compute the other group-bys in the cube, we collapse one dimension from a given G-tree at a time to construct a sub-G-tree and compute the corresponding group-bys of the sub-G-tree. For example, by collapsing dimension product in the original G-tree, given in Fig. 3, we get the subtree \{Month, \{\text{-Product}\}, \text{SalesMan}, \text{City}, \text{Customer}\}, shown in Fig. 5.

Fig. 4 shows the (sub)G-trees constructed for computing the data cube on dimensions A, B, C, and D. Each node in Fig. 4 represents a G-tree and the set of all corresponding group-bys. The ABCD tree at the top is constructed by one scan of the data set, and the corresponding group-bys (A, B, C, D), which are the (A, B, C), (A, B), (A), and () trees are computed during this scan. The sub-G-trees of the ABCD tree, which are the (\text{-A}B\text{CD}), (\text{-B}C\text{D}), and (\text{-A}B\text{-C})\text{D} trees, are formed by collapsing dimensions A, B, and C, respectively. The CD tree is a subtree of the BCD tree and recursively is also a subtree of the ABCD tree. The dimensions after “/” in the nodes denote common prefix dimensions for the tree at the node and all of its subtrees. A is the common prefix dimension for the ACD tree and its subtrees. All group-bys that are computed on the ACD tree and its subtrees form subcubes for A values Cube(\text{CD})\mid a_i. The leaf nodes originating from \text{a}_i are the MSPs for \text{Cube(\text{CD})}\mid a_i. In the sub-G-tree construction process, we also compute the bounds and use them for pruning, as described in the observation below.

**Observation 4.** With top-down aggregation, given a G-tree G with n dimensions \text{A}_1, \ldots, \text{A}_n, and a subtree \text{G}_k, by collapsing a dimension \text{A}_k, 1 < k < n, \text{A}_1, \ldots, \text{A}_{k-1} are the common prefix dimensions for \text{G}_k and all its subtrees. Each node (\text{a}_1, \ldots, \text{a}_{k-1}) of G gives the prefix dimension values for Cube(\text{A}_1, \ldots, \text{A}_n)\mid a_1, \ldots, a_{k-1}. The core of the cube consists of the MSPs corresponding to the leaf nodes of the branches originating from the node (\text{a}_1, \ldots, \text{a}_{k-1}). If the bounds of Cube(\text{A}_1, \ldots, \text{A}_n)\mid a_1, \ldots, a_{k-1} fail the given constraint, then those branches can be pruned.

**Example 9.** Consider the G-tree G in Fig. 3 as the original tree and the G-tree \text{G}_p in Fig. 5 as the subtree. Month is the prefix dimension for the group-bys on \text{G}_p, and the subtree of \text{G}_p. The subcubes are

9. It should be pointed out that a sub-G-tree is not part of an original G-tree but is obtained by collapsing a dimension of the original G-tree.

---

Fig. 4. Top-down bound prune cubing of Cube(ABCD): For each node, we show a G-tree (before “\{\}”), the group-bys (after “\{\}”) computed in the G-tree, and the shared dimensions (after the “\{\}”). The numbers show the order in which trees are built.

In G, the leaf nodes of the March subtree are the MSPs of Cube(\text{SalesMan}, \text{City})\mid \text{March}. With the constraint “Avg(Sale) in [15, 20],” \text{G} is pruned in constructing the subtree \text{G}_p. By following Table 6, the bounds for Cube(\text{SalesMan}, \text{City})\mid \text{March} are computed from the three leaf nodes originating from node (\text{March}) of G:

\[
\text{Avg(Cube(Sale)|Mar} = \max\{100/40, 100/20, 100/10\} = 10.
\]

\[
\text{Avg(Cube(Sale)|Mar} = \min\{100/40, 100/20, 100/10\} = 2.5.
\]

As [2.5, 10] violates “Avg(Sale) in [15, 20],” all three branches originating from (\text{March}) are removed from further computation. Similarly, Avg(Sale) of Cube(\text{SalesMan}, \text{City})\mid \text{January} is bounded as [40, 40], which violates the constraint, and is pruned. Avg(Sale) of Cube(\text{SalesMan}, \text{City})\mid \text{April} is bounded as [12.5, 25], which does not violate the constraint, and the branches from (\text{April}) are kept in \text{G}_p. The nodes in dashed lines in Fig. 5 highlight the fact that the branches from (\text{March}) and (\text{January}) are pruned before the \text{G}_p tree is generated, and importantly, they are permanently pruned from all future recursive computation.

### 7.3 The Top-Down Bound-Prune Cubing Algorithm

In this section, we present the Top-Down Bound-Prune Cubing (BP-Cubing(TD)) algorithm, shown in Algorithm 1, for computing iceberg data cubes. It performs bound-pruning in the top-down multway aggregation manner, and it uses the G-trees.

**Algorithm 1. The Top-Down Bound-Prune Cubing Algorithm**

**Input:** A data set D over dimensions \text{A}_1, \ldots, \text{A}_n and aggregation constraint \text{C}, assumed global.

**Output:** An n-dimensional iceberg cube on D satisfying \text{C}.

1. Build the G-tree T(\text{A}_1, \ldots, \text{A}_n) from D;
2. Output aggregates satisfying \text{C}, computed when T was built;
3. for \text{i} = 1 \ldots (n-1), do
4. BP-Cubing(T(\text{A}_1, \ldots, \text{A}_n), \text{A}_i);
Fig. 6. The G-tree for the data set in Table 1, with header table and side links.

// $B_i$, the $i$th dimension of $T$ is the dimension to be collapsed

Procedure BP-Cubing ($T(B_1, \ldots, B_k, B_i)$)

(5) $T_s \leftarrow \text{nil}$;
   // construct subtree $T_s$ by collapsing $B_i$.

(6) for each node $n_{i-1}$ of dimension $B_{i-1}$ in $T$, do

(7) $V_{i-1} \leftarrow$ dimension values on the path from root to $n_{i-1}$;
   // the MSPs for $\text{Cube}(B_1, \ldots, B_k)$ at $V_{i-1}$

(8) Let $M$ be the set of leaves originating from $n_{i-1}$;

(9) Compute the bounds for $\text{Cube}(B_1, \ldots, B_k)$ at $V_{i-1}$
   from $M$;
   // prune the paths if the bounds violate $C$

(10) if the bounds do not violate $C$ then

(11) Collapse $B$ and add the paths passing $n_{i-1}$ of $T$
   to $T_s$;

(12) Output aggregates on $T_s$ that satisfy $C$;

(13) for $j = (i + 1) \ldots (k - 1)$ do

(14) BP-Cubing($T_s(B_1, \ldots, B_{i-1}, B_{i+1}, \ldots, B_k), B_j$)

Let $D$ be a data set with $n$ dimensions $A_1, \ldots, A_n$ and
$C$ be the iceberg constraint. To compute the iceberg cube, first,
a G-tree $T$ is built by one scan of $D$ (line 1). As mentioned in
Observation 2, we simultaneously aggregate $n$ group-bys,
namely, $(A_1), (A_1, A_2), \ldots, (A_1, A_2, \ldots, A_n)$, when building
the G-tree. Those groups that satisfy $C$ are output (line 2).

The Bound-Prune Cubing (BP-Cubing) procedure computes
the iceberg cube by recursively collapsing dimensions
and building sub-G-trees. Given a G-tree, before a subtree is
built, bounds for branches are computed, and those branches
whose bounds fail the constraint are pruned from further computation (lines 9-11). The leaf nodes of a G-tree
are the MSPs for the G-tree, and these MSPs are used to
compute the corresponding aggregate bounds. $T_s$ is a
subtree of $T$, obtained by collapsing dimension $B_i$; the
collapsing computation is described in lines 6-11. Aggregates
are computed during the construction of $T_s$. Then, BP-
Cubing is recursively called (lines 13-14), where group-bys
of dimensions after $B_i$ are computed.

7.4 Bottom-Up Bound-Prune Cubing on G-Trees

The BP-Cubing(BU) algorithm is similar to Algorithm 1.
With the bottom-up cubing strategy, a group is computed
before its subgroups. There are several differences concerning
1) the G-tree used, 2) how sub-G-trees are constructed,
and 3) how bounding with MSPs are done. We describe these below.

We first consider the G-tree used. In BUC, a G-tree will
also contain a header table and side links. A header table
entry represents a one-dimensional group together with the
Corresponding aggregates, and it is linked as nodes with the
corresponding dimension value. Fig. 6 shows such a G-tree
for the sample Sales data set. The side-links are used to
efficiently derive aggregates of other groups in subsequent computation.

We now consider how sub-G-trees are constructed. In the
first G-tree for a data set, the header table contains
one-dimensional groups. In the sub-G-trees constructed from
a given G-tree, the header table contains groups whose
dimensionality is one level higher than that of groups for
the given G-tree. For the G-tree in Fig. 6, we build sub-G-
trees for the 2D subgroups of the $\text{City}$ groups, namely,
the groups of $(\text{Salesman, City})$, $(\text{Product, City})$, and $(\text{Month, City})$. Similarly, we build sub-G-trees for the
3D subgroups of the $(\text{Salesman, City})$ groups, and so on. The sub-G-trees are obtained by merging paths in the
original G-tree. For example, to compute the 2D subgroups
of $\text{Sydney}$, the subtree in Fig. 7 is constructed by merging
the paths from the root to the nodes on the $\text{Sydney}$ side link. In this process, the aggregates for the nodes above the
$\text{Sydney}$ nodes need to be modified to remove the
contribution of the non-$\text{Sydney}$ cities. For example, the
node (John) has the aggregates of $(30, 200)$ for both $\text{Perth}$ and $\text{Sydney}$ but only has the aggregates of $(10, 100)$ for
$\text{Sydney}$.

We now turn to bounding with MSPs. Recall that, with
the top-down aggregation strategy, all MSPs for a data
cube are “conveniently available” for bounding (by “con-
veniently available,” we mean that they are available
without extra overhead). In the bottom-up aggregation,
the “conveniently available” MSPs are those pointed to by
side links of certain header table entries. The observation
below describes how bounds are computed, and pruning is
achieved.

Observation 5. For a G-tree of dimensions $A_1, \ldots, A_n$, the leaf
nodes are the MSPs for $\text{Cube}(A_1, \ldots, A_n)$. With the BUC
strategy, for $a_k \in \text{domain}(A_k)(1 \leq k \leq n)$, the subgroups
of $a_k$ to be computed from the branches on the side link of $a_k$
comprise $\text{Cube}(A_1, \ldots, A_{k-1}) a_k$, and MSPs of the cube are
the leaf nodes on the side link for $a_k$ (available for bounding).
The subgroups from the side link of $a_k$ can be pruned if the
bounds of $\text{Cube}(A_1, \ldots, A_{k-1}) a_k$ fail the given constraint.

Example 10. With $\text{Cube}(\text{Month, Product, Salesman})|\text{Sydney}$,
shown in Fig. 6, its MSPs are the leaf nodes $(\text{Sydney}, 10, 100)$
and $(\text{Sydney}, 5, 100)$ on the side link for $\text{Sydney}$. Thus,
Algorithm 2. The Bottom-Up Bound-Prune Cubing (BU) Algorithm

Input: A data set \( D \) over dimensions \( A_1, \ldots, A_n \) and aggregation constraint \( C \).

Output: The \( n \)-dimensional iceberg cube on \( D \), satisfying \( C \).

1. Build the G-tree \( T(A_1, \ldots, A_n) \) from \( D \);
2. Output all aggregates in \( T(A_1, \ldots, A_n).\text{Header} \);
3. foreach \( a \in T.\text{Header} \) do
4. \( \text{BP-Cubing}(T(A_1, \ldots, A_n), \{ a \}); \)

Procedure BP-Cubing \( (T(B_1, \ldots, B_k), S) \)

// \( S \) is a set of dimension values as the condition for subcubes.

1. \( T_o = \text{nil} \);
2. Suppose \( a \) is on the \( i \)-th dimension of \( T \);
3. Let \( M \) be the nodes following the side link of \( a \);
4. Compute the bounds for \( \text{Cube}(B_1, \ldots, B_{k-1})|S \) from the leaves originating from \( M \);
5. if the bounds do not violate \( C \), then // pruning // Section 7.4
6. \( \text{construct the subtree } T_i(B_1, \ldots, B_{k-1}); \)
7. Output aggregates on \( T_i.\text{Header} \) that satisfy \( C \);
8. foreach \( a_i \in T_i.\text{Header} \) do
9. \( \text{BP-Cubing}(T_i(B_1, \ldots, B_{k-1}), S \cup \{a_i\}); \)

7.5 Interaction with Antimonotone Constraints

Our discussions have focused on complex nonantimonotone aggregation constraints. Antimonotone constraints can be easily incorporated in BP-Cubing as follows: Suppose an antimonotone constraint \( C' \) is present, in addition to the nonantimonotone constraint \( C \). First, in lines 2 and 12 of Algorithm 1, a group \( g \) is checked against both \( C \) and \( C' \) before being output. More importantly, before the bounds are computed at line 9, the branches of \( T \) originating from \( n_i \) are pruned from further computation if \( n_i \) fails \( C' \), as all subgroups of \( n_i \) will also fail \( C' \). With top-down cubing, the prefix groups before the collapsing dimension that fail \( C' \) are pruned. With bottom-up cubing, header-table entries that fail \( C' \) are pruned from further computation.

8 Experiments

In this section, we evaluate the performance of both the BP-Cubing(TD) and BP-Cubing(BU) algorithms with experiments. We compare the performance of these algorithms with that of the DnA algorithm [14], [15] on non-antimonotone aggregation constraints and also with that of the BUC algorithm [4] on antimonotone constraints.

DnA is recent work on pruning for nonantimonotone aggregation constraints. As DnA is not designed for pruning for antimonotone constraints, extra processing is needed to prune for such constraints. On the other hand, BUC uses the same partition-based bottom-up aggregation strategy as DnA and is designed for antimonotone constraints. Thus, BP-Cubing is compared with BUC on pruning with antimonotone constraints.

To do fair comparison, all algorithms were implemented with all possible optimization techniques. BUC was implemented with the collapsing duplicates optimization [4]. We added collapsing duplicates and indexing tuples to DnA, even though such optimizations were not reported in the original papers [14], [15]. Block memory allocation is used in the BP-cubing algorithms to reduce the number of calls of the dynamic memory allocation functions. It is assumed that, for all algorithms, data structures used can fit into memory. All experiments were performed on a PC with an i686 processor running GNU/Linux. As outputting the groups can take a significant amount of time for big data cubes, we choose to exclude the time for output in the timing for the algorithms. The thresholds for constraints are selected in a way such that there are at least 10 groups in the output. We experimented with constraints involving the aggregate functions Count, Avg, Var, and Sum.

8.1 Data Sets

We used real-world, as well as artificial, data sets in our experiments. When selecting the data sets, we considered the following data characteristics: dense versus sparse and random versus skewed. Table 7 summarizes the data sets used in our experiments.

The US census data set\(^{10} \) was collected in a 1990 US households survey. The original data set had 61 attributes such as \texttt{hwork1} (hours worked last week), \texttt{nchild} (number of own children on the household), and \texttt{valueh} (value of house). We selected 12 discrete attributes as

\(^{10}\text{ftp://ftp.ipums.org/ipums/data/ip19001.Z} \)
dimensions and a numerical attribute as the measure. The data set is dense and skewed.

The Weather data set\textsuperscript{11} contains real weather reports from various weather stations in 1985. We used these nine attributes as dimensions: station-id, longitude, solar-altitude, latitude, present-weather, weather-change-code, day, hour, and brightness. Their cardinalities, respectively, are 6,505, 351, 179, 152, 99, 10, 8, 3, and 2. We randomly generated values between 1 and 100 as the measure. This Weather data set is the same as that used in the experiments in [4], where BUC was shown to be efficient for sparse data.

The TPC-R data set\textsuperscript{12} is an artificial data set provided by the Transaction Processing Council and is designed for testing the performance of representative complex queries in high-level business decision-making environment. Its dimensions include customer, supplier, order, and shipment. The original TPC-R data set consists of several relational tables. We constructed a joined relation as our multidimensional data set. A numerical attribute is selected as the measure. The TPC-R data set is relatively dense and random.

### 8.3 BP-Cubing versus DnA on “Avg(X) in [δ₁, δ₂]”

In this set of experiments, we compare the performance of the BP-Cubing algorithms against DnA on the nonantimonotone constraint “Avg(X) in [δ₁, δ₂].”

Fig. 10 shows that the BP-Cubing algorithms scale very well when the [δ₁, δ₂] range threshold becomes looser. In all data sets, the BP-Cubing algorithms show modest linear increase in computation time. In contrast, the performance of DnA degrades significantly (which may be due to the fact that its pruning is at the tuple record level): When many groups need to be processed, the search cost for the minimal partition to approximate a group becomes high (see Section 9 for more discussions).

Figs. 10a and 10b show that in the dense data sets Census and TPC-R, the BP-Cubing algorithms significantly outperform DnA, and the BP-Cubing algorithm shows the best performance. For the constraint “Avg(X) in [500, 50,000]” over Census, DnA finishes in 65.04 seconds, whereas BP-Cubing(TD) finishes in only 3.53 seconds, which is 18.42 times faster. At the lower end, for the constraint “Avg(X) in [500, 10,000],” BP-Cubing(TD) is 7.09 times faster. In TPC-R, BP-Cubing(TD) outperforms DnA by 6.72-6.22 times. BP-Cubing(BU) also outperforms DnA overall, especially for larger constraint ranges. For the constraint “Avg(X) in [500, 50,000]” over Census, BP-Cubing(BU) is 7.09 times faster than DnA.

Fig. 10c shows that in the sparse Weather data set, BP-Cubing(BU) achieves more significant efficiency improvement over DnA than BP-Cubing(TD). In this data set,
the figure may suggest that the improvement of the BP-Cubing algorithms over DnA is not as pronounced as in Census and TPC-R. However, Fig. 9 has shown that in the sparse Weather data, the partition-based aggregation strategy of DnA is more efficient than the tree-based aggregation strategy of BP-Cubing. With an aggregation strategy that works not that well, the BP-Cubing algorithms still achieve better performance than DnA. Such dramatic result can only be attributed to the effectiveness of bound pruning.

As experiments have shown that BP-Cubing(TD) has better performance than BP-Cubing(BU) in general, BP-Cubing(TD) is used in later experiments for comparing with DnA.

8.4 BP-Cubing versus DnA on “Var(X) ≥ α”

In this section, we compare our bounding techniques against DnA on the constraint “Var(X) ≥ α”, which involves the complex nonmonotone function Var. For BP-Cubing, we obtain the upper bound for Var by following Table 6. For DnA, we derive the upper bound for Var as follows ([14, Example 4.2]):

$$\frac{QSUM(X)}{\text{Count}(X)} = \left( \frac{\text{psum}(1) \text{Count}(X)}{\text{psum}(2) \text{Count}(X)} \right)^2 + 2 \times \frac{\text{psum}(2) \times \text{nsum}(2)}{(\text{Count}(4))}\),$$

whose notations are explained later in Section 9.

The runtime of both BP-Cubing(TD) and DnA is shown in Figs. 11a, 11b, and 11c. When the Var threshold gets larger, the iceberg cubes get smaller, and both algorithms finish faster. BP-Cubing is always faster than DnA at all Var thresholds. The speedup is usually around several times. This can be attributed to the tighter bounds obtained using MSPs.

8.5 BP-Cubing versus DnA on “Sum(X) ≥ β”

We now compare BP-Cubing with DnA on computing iceberg cubes defined by “Sum(X) ≥ β.” As shown in Table 2, Sum(X) is a representative for our general bounding theory, which does not involve any optimization. The upper bound of Sum(X) needs to be computed for pruning. In BP-Cubing, this is computed following Table 2. In DnA, this is computed following [15] (see Section 9). The artificial data sets TPC-R and Weather contain only positive measure values, which renders Sum(X) ≥ β an antimono-

To ascertain the contribution of pruning toward the efficiency gain of BP-Cubing over DnA, for each algorithm, we collected the number of groups that are examined for pruning, namely, groups whose aggregates are bounded (approximated) and tested against the given constraint. There are two types of examined groups: a “true positive” (or nonsolution group) is a group whose approximated aggregate and its real aggregate pass the constraint, and a
would first rewrite the original aggregate into two subspaces of positive and negative measure values, respectively, so that a given constraint can be rewritten using antimonotone or monotone constraints in subspaces. For example, consider a measure \( X \), which can be positive or negative. Given the constraint \( \text{Avg}(X) \geq \alpha \), the authors would first rewrite the original aggregate into

\[
\text{Avg}(X) = \frac{\text{psum}(X)}{\text{Count}_1(X)} - \frac{\text{nsum}(X)}{\text{Count}_2(X)},
\]

where \( \text{psum} \) and \( \text{nsum} \) are the (absolute) sum of positive/negative \( X \) values, and \( \text{Count}_1 \) and \( \text{Count}_2 \) are rewriting \( \text{Count} \) in the positive and negative spaces. Then, they would use \( \frac{\text{psum}(X)}{\text{Count}_1(c)} - \frac{\text{nsum}(c)}{\text{Count}_2(c)} \) as a weaker antimonotone approximator for \( \text{Avg}(X) \), where \( c \) is some smallest subpartition of the given partition. If the approximate fails the threshold \( \alpha \), then all groups that are subgroups of \( X \) and supergroups of \( c \) are pruned.

BP-Cubing differs from DnA in several aspects:

1. Rather than the separately monotone rewriting strategy of DnA, in BP-Cubing, rewriting follows the principles that an algebraic aggregate function can be expressed as an algebraic expression of distributive aggregate functions and that the aggregate value for a group can be computed from the aggregates of MSPs. For example, given measure \( X \) with MSPs \( X_1, \ldots, X_n \), \( \text{Avg}(X) \) is rewritten into

\[
\frac{\text{Sum}(X)/\text{Count}(X)}{\text{Sum}(\text{Sum}(X))/\text{Sum}(\text{count}(X))}.
\]

2. In BP-Cubing, the optimization for complex aggregate functions such as \( \text{Avg} \) and \( \text{Var} \) can produce very tight bounds. Continuing with the previous example, by following Table 6, the optimized upper bound for \( \text{Avg}(X) \) is \( \text{Max}\{\text{Avg}(c_i)\} \), where \( c_i \) iterates over the smallest partitions. As \( \text{Avg}(c_i) \) is the real aggregate of a group in the search space, \( \text{Max}\{\text{Avg}(c_i)\} \) is the optimal approximator and is tighter (smaller) than

\[
\frac{\text{psum}(X)/\text{Count}_1(c)}{\text{Sum}(c)/\text{Count}_2(X)},
\]

which is the bound derived by DnA.

3. MSPs are nonempty groups of tuples and BP-Cubing prunes groups. Moreover, the G-tree structure of BP-Cubing greatly facilitates top-down multiway aggregation and saves computation, especially for dense data. DnA calculates aggregate bounds from tuples and prunes tuples—the bottom-up recursive partitioning aggregation strategy can incur extra cost searching for and pruning tuples that do not occur in any groups that satisfy a given constraint.

4. Extra processing is needed in DnA to incorporate antimonotone constraints such as \( \text{Count}(*) \geq n \). Continuing with the previous example, consider the constraint \( \text{Avg}(X) \geq \alpha \) and \( \text{Count}(*) \geq n' \) and the \( (ab) \) partition with \( c, d, e \) as further partitioning dimension values. In DnA, with recursive partitioning, even if \( (e) \) is infrequent, subpartitions of \( (ab) \) and \( (e) \), namely, \( (abe) \), \( (abce) \), \( (abcd) \), and \( (abcde) \), are not pruned (the Rollback tree was proposed to prune these groups). Note that these are only some of the subpartitions of \( (e) \) that should be pruned. In BP-Cubing, as discussed in Section 7.5, all subpartitions of a partition failing an antimonotone constraint are pruned.

The top-\( k \) average technique [7] was designed specifically for the constraint \( \text{Avg}(X) \geq \alpha \) and \( \text{Count}(*) \geq k \) and is not a general pruning technique for iceberg cubing. The tree-based bottom-up aggregation strategy was first proposed in [7]. Top-down aggregation was proposed independently in [16] for performing multiway aggregation in cubing; however, the work was focused on how to incorporate bottom-up pruning for antimonotone aggregation constraints into top-down aggregation. The BUC algorithm was proposed in [4], where pruning with antimonotone constraints was performed in the bottom-up recursive partitioning aggregation framework.

Our study is also related to constraint data mining [2], [3], [8], [9], [12], [13]. Agrawal and Srikant [2] wrote the first paper on pruning with the antimonotone constraint \( \text{Count}(*) \geq \alpha \) for association mining. In other works on...
association mining, pruning strategies with specific constraints were proposed, namely, minimum improvement constraints [3], succinct constraints [8], convertible constraints [9], item constraints [12], and support constraints [13]. These constraints are orthogonal to the aggregation constraints that we consider in iceberg cubing.

10 CONCLUSIONS
In computing iceberg cubes, pruning with nonantimonotone aggregation constraints has been a challenging problem. In this paper, we have proposed pruning with nonantimonotone constraints by estimating their upper and lower bounds on a data cube from aggregates of the most specific (or minimal) partitions. We have proposed iceberg cubing algorithms, called BP-Cubing, which incorporate bounding and pruning, and incorporated top-down and bottom-up aggregation strategies by using the G-tree structure. Our extensive experiments on real and artificial data sets have shown that BP-Cubing is effective for pruning under various data characteristics, and our iceberg cubing algorithms significantly outperform state-of-the-art iceberg cubing algorithms.

ACKNOWLEDGMENTS
The authors thank the reviewers for their helpful comments. A significantly abridged version of this paper appeared in [5].

REFERENCES

Xiuzhen Zhang received the PhD degree from the University of Melbourne. She is currently a senior lecturer in the School of Computer Science and Information Technology, RMIT University. Her research interests include database technology, data mining, machine learning, and their applications in new areas. She is a member of the ACM.

Guozhong Dong received the PhD degree from the University of Southern California in 1988. He is a professor at Wright State University. He previously taught at the University of Melbourne and Flinders University. He has served on numerous program committees including the IEEE International Conference on Data Engineering (ICDE), the IEEE International Conference on Data Mining (ICDM), the International Conference on Database Theory (ICDT), the ACM Symposium on Principles of Database Systems (PODS), the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, and the International Conference on Very Large Data Bases (VLDB). He was a program committee cochair of the 2003 International Conference on Web-Age Information Management (WAIM), is on the steering committee of the same conference series, is on the editorial board of the International Journal of Information Technology, and will serve as a program committee cochair of the 2007 Joint International Conference of APWeb and WAIM. His main research interests are databases, data mining, and bioinformatics. His research has been funded by government agencies such as the US National Science Foundation (NSF), the Australian Research Council (ARC), and the US Air Force Research Laboratory (AFRL) and private corporations. He has published more than 100 articles in journals and conferences; holds three US patents, and was a recipient of the IEEE ICDM 2005 Best Paper Award. He is a senior member of the IEEE and member of the ACM.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.