Multiway pruning for efficient iceberg cubing

Xiuzhen Zhang & Pauline Lienhua Chou

School of CS & IT, RMIT University, Melbourne, VIC 3001, Australia
{zhang,lchou}@cs.rmit.edu.au

Abstract. Effective pruning is essential for efficient iceberg cube computation. Previous studies have focused on exclusive pruning: regions of a search space that do not satisfy some condition are excluded from computation. In this paper we propose inclusive and anti-pruning. With inclusive pruning, necessary conditions that solutions must satisfy are identified and regions that can not be reached by such conditions are pruned from computation. With anti-pruning, regions of solutions are identified and pruning is not applied. We propose the multiway pruning strategy combining exclusive, inclusive and anti-pruning with bounding aggregate functions in iceberg cube computation. Preliminary experiments demonstrate that the multiway-pruning strategy improves the efficiency of iceberg cubing algorithms with only exclusive pruning.

1 Introduction

Since the introduction of the CUBE operator [6], the computation of data cubes has attracted much research [3, 7, 10]. Data cubes consist of aggregates for partitions produced by group-by's of all subsets of a given set of grouping attributes called dimensions. Given an aggregation constraint, aggregates in a data cube that satisfy the constraint form an iceberg cube [3]. Pruning is critical to the efficiency of iceberg cube computation, as the cube size grows exponentially with the number of dimensions. With traditional pruning strategies [1, 3, 4, 7, 9, 10] regions of the search space that do not satisfy a constraint are identified and then pruned from computation.

In this paper, we propose two new pruning techniques. (1) Inclusive pruning identifies conditions that the solutions must meet, and units under search that do not meet any of these conditions can not be solutions and thus are pruned. (2) Anti-pruning identifies regions where all units in the regions are solutions and testing for pruning is saved. We also propose inclusive and anti-pruning strategies with bounding aggregate functions for efficient iceberg cubing. Our initial experiments confirm that multiway-pruning improves the efficiency of cubing.

With traditional pruning, conditions are identified that warrant non-solutions. In this sense, it is termed *exclusive pruning*. With traditional exclusive pruning, the larger the solution set, the more tests for the pruning condition result in fruitless effort and become extra unnecessary cost. To the contrary of exclusive pruning, the cost for inclusive pruning does not increase and anti-pruning becomes better with larger solution set. Exclusive, inclusive and anti-pruning are

Citation:
Zhang, X and Chou, P 2006, 'Multiway pruning for efficient iceberg cubing', in S. Bressan, J. Kung & R. Wagner (ed.) Proceedings of the 17th international conference on database and expert systems applications (DEXA 2006), Heidelberg, Germany, 21 September 2006.

collectively called the *multiway pruning* strategy. Without inclusive pruning and anti-pruning, traditional exclusive pruning still compute the complete solutions and prunes correctly. However, it incurs extra cost. Inclusive and anti-pruning can not replace exclusive pruning, but complement exclusive pruning and in many cases greatly reduce the cost for pruning.

1.1 Related work

Studies in the iceberg cubing literature have all focused on exclusive pruning strategies in iceberg cube computation [3, 7, 10, 12].

Another related area is constraint data mining. Several types of constraints have been proposed and their properties are used for pruning [1, 8, 9]. All these work focus on exclusive pruning, expect in Reference [8]. The succinct constraints in Reference [8] are conceptually similar to inclusive pruning, however they are constraints orthogonal to the constraints for exclusive pruning. Our inclusive pruning does not rely on extra constraints and it is used to enhance exclusive pruning.

The term *inclusive pruning* was coined by Webb [11] in the area of machine learning. Specific pruning rules were developed for classification learning. We introduce inclusive pruning into iceberg cube mining and develop inclusive pruning strategies for aggregation constraints based on bounding.

The concept of border was used in data mining to represent large spaces [2, 5]. Rather than computing borders, we focus on using borders for more effective pruning and to compute all solutions in the space represented by borders.

2 Preliminaries on data cubes

Dimensional datasets are defined by *dimensions* and *measures*. In Table 1, there are 4 dimensions Month, Product, SalesMan (Man) and City, and Sale (aggregated as Sum(Sale)) is the measure. Throughout this paper we use upper-case letters to denote dimensions and lower-case letters to denote dimension-values. For example $(\mathtt{A}, \mathtt{B}, \mathtt{C})$ denotes a group-by and $(\mathtt{a}, \mathtt{b}, \mathtt{c})$ denotes a partition of tuples from the $(\mathtt{A}, \mathtt{B}, \mathtt{C})$ group-by, and is called a *group*. We also use an upper case letter for a measure attribute to represent the set of values for the measure.

The group-bys in a data cube form a lattice structure called the *cube lattice*. Fig. 1 shows an example 4-dimensional cube lattice, where each node denotes a group-by list. The empty group-by that aggregates all tuples in a dataset is at the bottom of the lattice and the group-by with all dimensions is at the top of the lattice. Group-bys form subset/superset relationships. Groups from different group-bys also form super-group/sub-group relationships.

A special value "*" is assumed in the domain of all dimensions, which can match any value of the dimension. With the Sales dataset, the $(\mathtt{Jan}, *, *, *)$ group denotes the partition of tuples with $\mathtt{Month} = \mathtt{January}$ and no restriction on values for the other dimensions. For simplicity, $(\mathtt{Jan}, *, *, *)$ is also written as $(\mathtt{Jan})$.

Table 1. A sales dataset, partially aggregated

Month	Prod	Man	City	Cnt(*)	Sum(Sale)
Jan	Toy	John	Perth	5	200
Mar	TV	Peter	Perth	40	100
Mar	TV	John	Perth	20	100
Mar	TV	John	Syd	10	100
Apr	TV	Peter	Perth	8	100
Apr	Toy	Peter	Syd	5	100

Table 2. The bounds of some aggregate functions ($\{X_i | i = 1..n\}$ are MSPs)

F	upper bound; lower bound
Cnt	$\sum_i \text{Cnt}(X_i)$; $\min_i \text{Cnt}(X_i)$
Max	$\max_i \text{Max}(X_i)$; $\min_i \text{Max}(X_i)$
Min	$\max_i \text{Min}(X_i)$; $\min_i \text{Min}(X_i)$
Avg	$\max_i \text{Avg}(X_i)$; $\min_i \text{Avg}(X_i)$
Sum	if there exists $\text{Sum}(X_i) > 0$, $\sum_{\text{Sum}(X_i)>0} \text{Sum}(X_i)$, otherwise $\max_i \text{Sum}(X_i)$; if there exists $\text{Sum}(X_i) < 0$, $\sum_{\text{Sum}(X_i)<0} \text{Sum}(X_i)$, otherwise $\min_i \text{Sum}(X_i)$

The 3-dimensional group (`Jan`, `TV`, `Perth`) is a sub-group of the 2-dimensional groups (`Jan`, `TV`), (`Jan`, `Perth`) and (`TV`, `Perth`).

Given a dataset S of n dimensions A_1, ..., A_n with the measure X, the data cube of applying F on S can be expressed in two ways: (1) Cube$(A_1, ..., A_n)$ can be viewed as the set of possible group-bys, or the set of possible groups for all group-bys. (2) Cube$(X) = \{X_i = \{t[X] \mid t \in g_i\} \mid g_i$ is a partition of the cube$\}$; that is, a data cube is expressed as partitions of multi-set of measure values following the grouping of tuples.

3 Bounding aggregate functions

Bounding was proposed as a technique to prune for complex aggregation constraints [12]. The idea of bounding is to estimate the upper and lower bounds for data cubes from the most specific partitions (MSPs) of data. The Sales dataset comprises 6 MSPs, as shown in Table 1. Each MSPs has been aggregated with functions Count(*) (denoted as Cnt(*)) and Sum(Sale).

We generalize the original definition of data cubes [6]. Fig. 1 shows Cube (`ABCD`), and it can be decomposed into two lattice structures: The nodes in the right polygon form Cube (`BCD`). For each partition of data with $\mathtt{a_i} \in$ domain(`A`), the nodes in the left polygon form a data cube on dimensions {`B, C, D`} conditioned on $\mathtt{a_i} \in$ domain(A). In later discussions we use the term data cube to refer to both unconditional data cubes and conditional data cubes.

The concept of data cube core was coined by Gray et al. [6] it was generalized to conditional data cubes. Given Cube$(A_1, ..., A_n)$ over measure X, all n-dimensional partitions $\{(a_1, ..., a_n) | a_i \neq *, a_i \in \text{domain}(A_i), 1 \leq i \leq n\}$ comprise the *core* of the data cube and each partition g in the core is a *Most Specific Partition* (MSP). The multi-set of measure values for tuples in an MSP g, namely $\{t[X] | t \in g_i\}$, is an MSP of the measure. All aggregates in a data cube can be

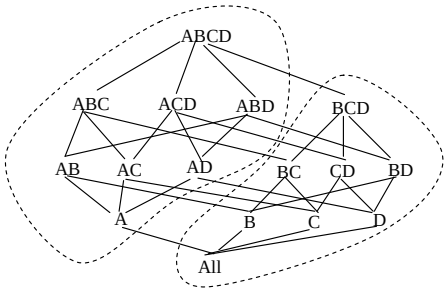


Fig. 1. Cube(ABCD) decomposition

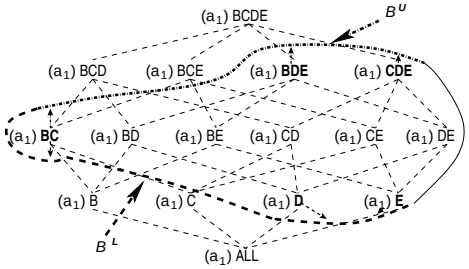


Fig. 2. *An anti-pruning border on the lattice conditioned on* (a_1).

computed from its MSPs. In Fig. 1, any group in $\texttt{Cube(BCD)}|\texttt{a}_1$ can be computed from some $(\texttt{a}_1, \texttt{b}_i, \texttt{c}_j, \texttt{d}_k)$ groups and any group in Cube(BCD) can be computed from some $(\texttt{b}_i, \texttt{c}_j, \texttt{d}_k)$ groups, where i, j and k iterates over values in the domain of B, C and D respectively.

Given an aggregate function F, and a multi-dimensional dataset with measure X, the upper (lower) bound for a data cube is a real number such that for any partition $X_i \in \text{Cube}(X)$, $F(X_i)$ is no larger (smaller) than the upper (lower) bound. An aggregate function F is boundable for a data cube if the upper and lower bounds for the data cube can be determined with a single scan of the local aggregate values of the MSPs. The bounds for some commonly seen aggregate functions are listed in Table 2. We use Count(*) as an example to explain. The Count(*) for any group in a data cube is no larger than $\text{Sum}_i \text{Cnt}(X_i)$, the sum of Count(*) all MSPs. The Count(*) for any group in a data cube is no smaller than $\text{Min}_i \text{Cnt}(X_i)$, the smallest Count(*) among all MSPs. Moreover, these bounds are the tightest bounds for Count(*). Indeed arithmetic expressions of base functions Sum, Count, Min, and Max are boundable. Details of bounding algorithms are discussed in Reference [12].

4 Inclusive pruning with bounding

Inclusive pruning is applied before computing the bounds of data cubes for exclusive pruning. Dimension values that solutions in a lattice must have are called *inclusive dimension values*. Groups that are defined by only non-inclusive dimension-values are definitely not solutions and should be pruned.

Definition 1. *Given a data cube to be computed with some constraint, the inclusive dimension values are those that define groups that are solutions.*

From this definition, dimension values that do not involve any solutions are non-inclusive dimension values.

Theorem 1. *(***Inclusive pruning***) Given a cube lattice* $\mathcal{L}$*, an aggregation constraint, and a set of inclusive dimension values, groups in* $\mathcal{L}$ *that are defined by only non-inclusive dimension values are not solutions and can be pruned.*

Proof. All solution groups in $\mathcal{L}$ must be grouped by at least one inclusive dimension value. A group in $\mathcal{L}$ whose grouping dimension values are solely defined by non-inclusive dimensions can not be a solution and so can be pruned.

The question to answer now is how to identify the inclusive dimension values in a cube lattice before it is computed. As will be seen in Section 6, the inclusive dimension values for a cube are decided in the previous computation of its super-cubes: all dimensions-values whose bounds have non-empty intersection with the interval defined by thresholds of the given constraint are inclusive dimension values for the cube. Importantly these bounds can be computed by a single traversal of the MSPs. As a result we have the benefit of pruning with little extra cost. Inclusive pruning is achieved by removing branches that contain only non-inclusive dimension-values. Note that these branches are pruned from all sub-trees that are to be computed.

5 Anti-pruning with bounding

When the selectivity of an aggregation constraint is high, a large number of groups are qualified as iceberg groups. When the data is skewed, the iceberg groups reside in a few regions of the cube. In both cases, examining the pruning conditions on these groups does not result in any pruning. We introduce the notion of the *Anti-pruning border* to mark regions where all groups are solutions.

Definition 1 (Anti-pruning region) *In iceberg-cubing, given a lattice $\mathcal{L}$ an anti-pruning region is a convex space of groups that are all solutions. An anti-pruning region $\mathcal{B}$ can be denoted as $\langle \mathcal{B}^L, \mathcal{B}^U \rangle$: $\mathcal{B}^L$ is the lower border where none of their supergroups are in $\mathcal{B}$. $\mathcal{B}^U$ is the upper border consisting of groups in $\mathcal{B}$ where none of their subgroups are in $\mathcal{B}$.*

Example 1 *An example anti-pruning border on the lattice of* B, C, D, *and* E *conditioned on (a_1) is shown in Fig. 2. The upper border consists of the (a_1BC), (a_1BDE), and (a_1CDE) group-bys and the lower border consists of the (a_1BC), (a_1D), and (a_1E) group-bys. The anti-pruning region contains groups that are supergroups of some group in the (a_1BC), (a_1BDE), and (a_1CDE) group-bys as well as subgroups of some group in the (a_1BC), (a_1D), and (a_1E) group-bys.*

An anti-pruning region is a maximum region in a group-lattice where pruning is unnecessary. The groups within the region covered by the anti-pruning borders are definitely solutions. We aim to compute anti-pruning regions that are lattices with little extra cost. We observe that lattices are convex spaces: Given a lattice conditioned on a group g, the lattice is a convex space where all groups are subgroups of g and supergroups of some MSPs. We make use of bounding to detect anti-pruning regions that are lattices, as shown in the observation below.

Observation 1 *Given a lattice conditioned on a group g, if both bounds satisfy a given constraint, the MSPs are the upper anti-pruning border $\mathcal{B}^U$, and g is the lower anti-pruning border $\mathcal{B}^L$; the lattice of g is an anti-pruning region. All groups in the lattice satisfy the constraint.*

For iceberg cubes with complex aggregation constraints, the cost of checking for the pruning conditions is high. Detecting a group-lattice that is an anti-pruning region supposedly improves the efficiency of cubing algorithms.

6 Multiway Bound-Prune Cubing on G-trees

The Group tree (G-tree) is our data structure for cubing. The G-tree for an n-dimensional dataset is of depth n, where each level represents a dimension. A common path starting from the root collapses the tuples with common dimension-values. Each tree node keeps the local aggregates necessary to compute the iceberg cube for a given aggregate function. A G-tree is constructed by one scan of the input data.

The G-tree in Fig. 3 represents the Sales dataset. To compute the data cube with aggregate function Average, the local aggregates in each node are Sum(`Sale`) and Cnt($*$). On the leftmost path node (`March`) shows that there are 70 tuples with Sum(`Sale`) = 300 in the $(\texttt{March}, *, *, *)$ group, whereas the node (`Peter`) shows that there are 40 tuples with Sum(`Sale`) = 100 in the $(\texttt{March}, \texttt{TV}, \texttt{Peter}, *)$ group.

6.1 Top-down aggregation on G-trees

Top-down aggregation on G-trees is based on the following observation: Construction of an n-dimensional G-tree has computed n group-bys, namely the group-bys whose dimensions are prefixes of the given list of dimensions. With the G-tree in Fig. 3, the root node has the aggregate for the group $(*, *, *, *)$ with Cnt $(*)$ = 88 and Sum (`Sale`) = 700; The nodes at level one compute the aggregates for groups in $(\texttt{Month}, *, *, *)$, which are $(\texttt{March}, *, *, *, 70, 300)$, $(\texttt{January}, *, *, *, 5, 200)$, and $(\texttt{April}, *, *, *, 13, 200)$. The nodes at the next 3 levels compute the aggregates for $(\texttt{Month}, \texttt{Product})$, $(\texttt{Month}, \texttt{Product}, \texttt{Man})$, and $(\texttt{Month}, \texttt{Product}, \texttt{Man}, \texttt{City})$ respectively.

Let A, B, C and D denote the four dimensions of the Sales dataset. The group-bys that are not represented on the G-tree in Fig. 3 are computed by collapsing one dimension at a time to construct sub-G-trees. In Fig. 4, each node represents a G-tree and all group-bys that are simultaneously computed on the G-tree. The `ABCD`-tree representing the tree of Fig. 3 is at the top. The group-bys `(A,B,C,D)`, `(A,B,C)`, `(A,B)`, `(A)` and `()` are computed on the `(A,B,C,D)`-tree. The sub-trees of the `ABCD`-tree, (−A)`BCD`, `A`(−B)`CD`, and `AB`(−C)`D`, are formed by collapsing on dimensions A, B, and C respectively.

6.2 Multiway pruning on G-trees

In Fig. 3, the leaf nodes of the G-tree are the MSPs for Cube $(\texttt{Mon}, \texttt{Prod}, \texttt{Man}, \texttt{City})$. The dimensions after "/" in each node of Fig. 4 denote prefix dimensions for the tree at the node and all its sub-trees. `A` is the prefix dimension for the `ACD`-tree and its sub-trees. All group-bys that are computed on the `ACD`-tree and its

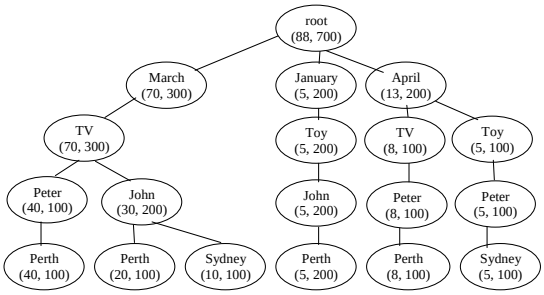


Fig. 3. A sample G-tree

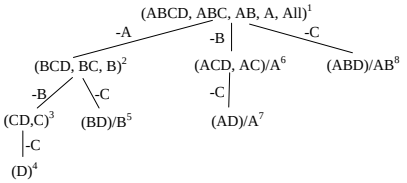


Fig. 4. Top-down cubing of a 4 dimensional data cube with shared dimensions

sub-trees form data cubes conditioned on some A-value, Cube(`CD`)|$\mathtt{a_i}$, where i iterates over the values of A. The leaf nodes originated from a_i are the MSPs for Cube(`CD`)|$\mathtt{a_i}$. The bounds for Cube(`ACDE`)|$\mathtt{a_i}$ are obtained by a traversal of the leaf nodes originated from $\mathtt{a_i}$. With the G-tree over dimensions `Month, Product, Man`, and `City` in Fig. 3, before collapsing on Product, we calculate the bounds for the cube lattices conditioned on each dimension value. Following Fig. 2, the bounds for Cube (`Product`, `Man`, `City`)|`March` are computed from the three leaf nodes originated from `(March)` of G:
$\underline{\text{Avg}(\text{Cube}\,(\mathtt{Man}, \mathtt{City})|\mathtt{Mar})} = \text{Max}(\{100/40, 100/20, 100/10\}) = 10$;
$\underline{\text{Avg}(\text{Cube}\,(\mathtt{Man}, \mathtt{City})|\mathtt{Mar})} = \text{Min}(\{100/40, 100/20, 100/10\}) = 2.5$.

Inclusive pruning is achieved by identifying $\mathcal{I}$, the set of inclusive dimension-values for a cube lattice before its aggregates are computed. Given a G-tree G on dimensions $A_1, ..., A_n$, the bounds for Cube$(A_{k+1}, ..., A_n)|a_1, ..., a_{k-1}$, $1 < k < n$, are computed from the leaf nodes originated from the node a_k. If the bound interval does not violate the given constraint, a_k should be added to $\mathcal{I}$. After $\mathcal{I}$ is obtained, the G-tree is traversed again in depth-first order for inclusive pruning. At each node, if its dimension-value is from $\mathcal{I}$, the traversal to its children is terminated because the branch is not to be trimmed. If the traversal reaches a leaf node, the branch is trimmed. From the leaf upwards, the nodes on the branch that have no children are deleted.

Observation 2 *(***Inclusive pruning***) Consider computing an iceberg cube with the constraint "$F(X)$ in $[\delta_1, \delta_2]$". Given a G-tree with n dimensions A_1, ..., A_n, Suppose $[b_1, b_2]$ are the bounds for* Cube$(A_{k+1}, ..., A_n)|a_1, ..., a_{k-1}$*, the sub-cube by collapsing a_k ($1 < k < n$), inclusive dimension-values are those a_k such that $[b_1, b_2] \cap [\delta_1, \delta_2] \neq \phi$. Branches defined by only non-inclusive dimension-values are pruned from computation.*

After inclusive pruning branches on a G-tree that can not contain solutions have been pruned from computation. With the bounds computed for each cube lattice, exclusive- and anti-pruning are then applied.

Algorithm 1 The MBPC Algorithm

// Assume that an n-dimensional G-tree T with aggregate function F is constructed.
// The iceberg cube is computed by calling MBPC($T(A_1, ..., A_n)$, ϕ).
Input: a) T is a G-tree with conditional base B.
b) Aggregate constraint " $F(X)$ in $[\delta_1, \delta_2]$" is assumed global.
Output: The iceberg groups in the iceberg cube.
MBPC($T(B_1, ..., B_m)$, B)
(1) **for each** dimension $i = 1..m-1$ **do**
(2) **for each** node b_i of dimension B_i on T
(3) Output the group g conditioned on B if F(g) in $[\delta_1, \delta_2]$;
(4) Compute the bounds $[b_1, b_2]$ and $\mathcal{I}$ for $Cube(B_{i+1}, ..., B_m)|B \cup \{b_i\}$;
// *inclusive pruning*
(5) Prune branches formed by only dimension-values from $\neg\mathcal{I}$;
(6) **if** $([b_1, b_2] \cap [\delta_1, \delta_2] = \phi)$
(7) break; // *exclusive pruning*
(8) **else if** $([b_1, b_2] \cap [\delta_1, \delta_2] = [b_1, b_2])$
(9) compute all groups in Cube$(B_{i+1}, ..., B_m)|B \cup \{b_i\}$;
(10) break; //*anti-pruning*
(11) for $j = i + 1..m$ do
(12) Construct the sub-tree T_s by collapsing B_j;
(13) **MBPC**($T_s, B \cup \{b_i\}$);

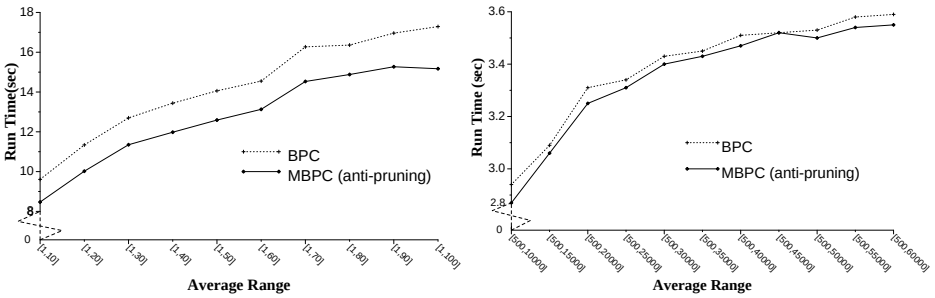
Observation 3 *(***Exclusive- and anti-pruning***) Consider computing an iceberg cube with the constraint "$F(X)$ in $[\delta_1, \delta_2]$". Given a G-tree for dimensions A_1, ..., A_n, let $[b_1, b_2]$ be the bounds for* Cube$(A_{k+1}, ..., A_n)|a_1, ..., a_{k-1}$*, the cube by collapsing A_k ($1 < k < n$). If $[b_1, b_2] \cap [\delta_1, \delta_2] = \phi$, the branches originated from the node $(a_1, ..., a_{k-1})$ can be pruned; otherwise if $[b_1, b_2] \cap [\delta_1, \delta_2] = [b_1, b_2]$, all groups in* Cube$(A_{k+1}, ..., A_n)$ *are qualified groups.*

6.3 The MBPC algorithm

We present the Multiway Bound-Prune Cubing (MBPC) algorithm for computing iceberg cubes, with inclusive- exclusive and anti-pruning strategies. The algorithm is shown as Algorithm 1.

A G-tree is built by one scan of the input dataset and is the input for MBPC. MBPC is a recursive procedure where with each recursion, an additional dimension-value is accumulated in g, the group to be computed. For the ease of presentation, one group g is processed a time in Algorithm 1. Indeed with top-down cubing on the G-tree, multiple groups are simultaneously processed. The steps involved in each group are the same.

At line 3 of the algorithm, the groups that are computed on an G-tree are conditioned on the conditional base B of the tree. Initially the conditional base is empty. The prefix dimension-value before the collapsing dimension is accumulated into the conditional base B within each recursion. The bounds are obtained



(a) Weather (100,000 tuples, 9 dimensions) (b) Census (88,443 tuples, 12 dimensions)

Fig. 5. Performance comparison of MBPC (anti-pruning) and BPC

when the branches under the prefix groups are amalgamated. No extra scan of the tree is required.

Inclusive pruning (line 5) is applied before further aggregation of subgroups. For the top-down iceberg cubing on the G-tree, a list $\mathcal{I}$ is created when the G-tree is built. The list consists of dimension-values of the G-tree, each is associated with a bit-vector. When the G-tree is traversed in depth-first order to obtain the bounds for the nodes on the G-tree, the bounds for the dimension-values in the the list are also obtained; they determine which branches can be trimmed.

The test for anti-pruning (line 8) can be combined with that for exclusive pruning (line 6) as a single test, where the intersection of the two ranges is performed only once. Anti-pruning (line 9) is applied by aggregation of groups without any bounding or bounding test being performed.

7 Experiments

We conducted preliminary experiments to examine the performance of MBPC. We implemented MBPC with anti-pruning and compared its performance with that of BPC, that uses only exclusive bound pruning. Experiments were conducted on a PC with i686 processor running GNU/Linux.

Two datasets were used in our experiments. The weather dataset [1] is the weather reports collected at various weather stations globally in 1985. Nine of the attributes such as `station-id`, `longitude`, and `latitude` are used as dimensions and the cardinalities range from 2 to 6505. Numbers between 1 and 100 were randomly generated as the measure. The US census dataset [2] was collected from surveys about various aspects of individuals in the households in US in 1990. The original dataset has 61 different attributes such as `hrswork1`(hours worked last week) and `valueh`(value of the house). We selected 12 discrete attributes as dimensions and a numeric attribute as the measure. Census is dense and skewed and Weather is very sparse.

[1] http://cdiac.ornl.gov/ftp/ndp026b/SEP85L.DAT.Z
[2] ftp://ftp.ipums.org/ipums/data/ip19001.Z

The results of computing iceberg cubes with the constraint "Avg(X) in $[\delta_1, \delta_2]$" are shown in Fig. 5. The time reported does not include time for output. Generally anti-pruning further improves the efficiency of the BPC algorithm. Moreover, as δ_2 get larger and the iceberg cube to compute get larger, more pronounced improvement can be observed. MBPC shows significant improvement in run time over BPC on Weather. MBPC outperforms BPC consistently by about 13% for different aggregation constraint thresholds. MBPC shows modest improvement over BPC on the Census dataset.

8 Conclusions

We have proposed multiway pruning strategies in iceberg cubing: exclusive, inclusive and anti-pruning. We have also presented an iceberg cubing algorithm with the multiway pruning strategy with bounding. Our initial experiments with anti-pruning has shown that the multiway pruning strategy can significantly improve efficiency of cubing algorithms, especially on computing large cubes. More extensive experiments are underway to examine the effectiveness of inclusive pruning and the multiway pruning strategy with respect to general data characteristics.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB'94*, pages 487–499, Santiago, Chile.
2. R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. of SIGMOD'98*, pages 85–93.
3. K Beyer and R Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. of SIGMOD'99*.
4. L. Chou and X. Zhang. Computing complex iceberg cubes by multiway aggregation and bounding. In *Proc. of DaWak'04 (LNCS 3181)*, Zaragoza Spain.
5. G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proc. of KDD'99*, pages 15–18, San Diego, USA.
6. J Gray et al. Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1), 1997.
7. J Han, J Pei, G Dong, and K Wang. Efficient computation of iceberg cubes with complex measures. In *Proc. of SIGMOD'01*.
8. R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. of SIGMOD'98*.
9. J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proc. of ICDE'01*.
10. K. Wang et al. Divide-and-approximate: A novel constraint push strategy for iceberg cube mining. *IEEE TKDE*, 17(3):354–368, March 2005.
11. G. I. Webb. Inclusive pruning: a new class of pruning rules for unordered search and its application to classification learning. In *Proc. of ACSC'96*.
12. X. Zhang, L. Chou and G. Dong. Efficient computation of iceberg cubes by bounding aggregate functions. *IEEE TKDE*, In submission.