Computing complex iceberg cubes by multiway aggregation and bounding

Chou, Pauline; Zhang, Xiuzhen

Chou, P., & Zhang, X. (2004). Computing complex iceberg cubes by multiway aggregation and bounding. Data Warehousing and Knowledge Discovery: Sixth International Conference, 108–117. https://doi.org/10.1007/b99817

Published Version: https://doi.org/10.1007/b99817

Computing Complex Iceberg Cubes by Multiway Aggregation and Bounding

LienHua Pauline Chou and Xiuzhen Zhang

School of Computer Science and Information Technology
RMIT University, Melbourne, VIC., Australia, 3000
{lchou, zhang}@cs.rmit.edu.au

Abstract. Iceberg cubing is a valuable technique in data warehouses. The efficiency of iceberg cube computation comes from efficient aggregation and effective pruning for constraints. In advanced applications, iceberg constraints are often non-monotone and complex, for example, "Average cost in the range [δ_1, δ_2] and standard deviation of cost less than β". The current cubing algorithms either are efficient in aggregation but weak in pruning for such constraints, or can prune for non-monotone constraints but are inefficient in aggregation. The best algorithm of the former, Star-cubing, computes aggregations of cuboids simultaneously but its pruning is specific to only monotone constraints such as "COUNT$(*)$ $\geq \delta$". In the latter case, the Divide and Approximate pruning technique can prune for non-monotone constraints but is limited to bottom-up single-group aggregation. We propose a solution that exhibits both efficiency in aggregation and generality and effectiveness in pruning for complex constraints. Our bounding techniques are as general as the Divide and Approximate pruning techniques for complex constraints and yet our multiway aggregation is as efficient as Star-cubing.

1 Introduction

Data Warehousing and OLAP technologies require summarised and subject-oriented views to data for better and faster high level analysis and decision making. To this purpose, data is modelled multi-dimensionally. In a multi-dimensional model, *dimensions* such as Product and Customer-Group describe the subjects of interest; the *measure* such as total sales is the target of analysis in terms of dimensions.

A data cube generalises the SQL Group-By operator [6] to compute Group-Bys of all combinations of dimensions. Each Group-By is a *cuboid* which comprises a set of groups grouped by the same dimensions. For example, (a_1,b_1, c_1,d_1) is one of the groups in the cuboid $ABCD$. Note that upper case letters denote dimensions and cuboids while subscripted lower-case letters denote dimension-values and groups. The cuboid-lattice in Figure 1 for the cube on dimension A, B, C, and D shows the parent and child relationship between cuboids. Since a parent cuboid has more grouping-dimensions than a child cuboid, a group in the parent cuboid is a sub-group of some group in the child cuboid. For example, ($a_1b_1c_1$) is a sub-group of (a_1b_1).

Citation:
Chou, P and Zhang, X 2004, 'Computing complex iceberg cubes by multiway aggregation and bounding', in Y Kambayashi et al. (ed.) Data Warehousing and Knowledge Discovery: Sixth International Conference, Berlin, 8 November 2004.

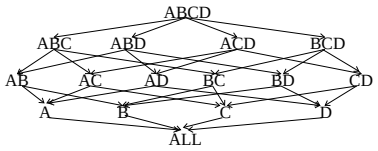

Fig. 1. *A cuboid lattice*

Given a dataset of k dimensions where each dimension has a cardinality of p, potentially the cube has $(p+1)^k$ groups. Given an aggregate function, cubing is to compute all groups in the data cube for answering queries on aggregations. SUM() COUNT() MIN() MAX(), and AVG() are common SQL aggregate functions. Cubing algorithms [11, 10, 3, 8] exploit the advantage of shared-computation by computing each child-cuboid from a parent cuboid. This aggregation strategy is refered to as *top-down* computation. The Multiway Array Cube [10] is the most efficient among these algorithms.

Iceberg cubes [2] were proposed to compute only interesting groups that meet users information need. Users specify *iceberg constraints* on the aggregate values of groups and only satisfying groups are computed. Iceberg constraints can be either *monotone* or *non-monotone* [2]. A monotone constraint possesses the property that if a group fails the constraint, so do all its sub-groups [1]. The monotone property is commonly used for pruning in iceberg cubing [2]. BUC [2] and H-cubing [7] are current iceberg cubing algorithms which apply the divide-and-conquer approach where groups are computed before their sub-groups. Such an aggregation strategy is called *bottom up* with respect to the cuboid lattice. Pruning for monotone constraints can be easily applied to the bottom up approach.

Star-cubing [4] is the most efficient iceberg cubing algorithm. It makes use of shared computation as in Multiway Array Cubing [10] and incorporates pruning for monotone constraints. It is shown that Star-cubing is more efficient than BUC and H-cubing. However, the optimisation in Star-cubing is specific to only monotone constraints such as "COUNT$(*) \geq \delta$".

The diversity of users' information needs means that practical constraints are often complex and non-monotone, such as constraints with AVG or SD (StandardDeviation). Iceberg cubing for complex non-monotone constraints is challenging due to the difficulty of pruning. Recently, Wang et al. [9] proposed an algorithm *Divide and Approximate* for pruning for non-monotone constraints. The pruning techniques however are only suitable for BUC like bottom-up single group oriented computation. A more detailed analysis of Divide and Approximate pruning is provided in Section 4.

In this work, we propose the iceberg cubing algorithm *Bound-cubing*, which incorporates bounding techniques for pruning with complex non-monotone constraints. It is not only general and effective in pruning but also well suited for an efficient top-down shared aggregation strategy. Importantly, little overhead is incurred by bounding. Our experiments show that *Bound-cubing* is marginally more efficient than Star-cubing in computing iceberg cubes with monotone con-

[1] In the literature, this is also called anti-monotone.

straints; it significantly outperforms the Divide and Approximate algorithm [9] for non-monotone constraints.

2 Bounding Aggregate Functions for Pruning

In this section, we describe our bounding techniques for pruning. We aim at pruning with general constraints of the form "$f(x)$ *op* δ", where $f(x)$ is an aggregate function, *op* is the comparison operator $\geq$ or $\leq$, and δ is a threshold. We will focus on bounding for a single constraint; this can be easily extended to multiple constraints. The main idea of bounding is to bound the aggregation values of a set of groups from the same values used for computing the aggregations of the groups. The derived bounds can be used to prune the set of groups at once.

In cube computation, aggregate functions are categorised into *Distributive*, *Algebraic*, and *Holistic* functions, based on how a group can be aggregated from its sub-groups [6].

- **Distributive**: An aggregate function F is distributive if for a group g, $F(g)$ can be aggregated from the values of $F(g_s)$, where g_s groups are g's sub-groups. Aggregate functions SUM, COUNT, MIN, and MAX are distributive.
- **Algebraic**: An aggregate function F is algebraic if for a group g, $F(g)$ can be computed from a known M number of intermediate aggregations of g's sub-groups by some aggregate functions. For clarity, we call these aggregate functions in F *local aggregate functions*. Aggregate functions AVG, SD, MaxN, MinN, and CenterOfMass are algebraic. Taking AVG as an example, AVG = "SUM(SUM(m))/SUM(COUNT($*$))", where m is the measure. Note that distributive functions are a special case of algebraic functions, where $M = 1$ with a single local aggregate funciton being F.
- **Holistic**: There is no fixed number of intermediate aggregate values that can aggregate a group for a holistic function. RANK is a holistic function. There exists no known computation methods other than computing the aggregation from the raw data hence they are not considered in this work.

Given a cube defined on k dimensions, the *data cube core* [6] refers to the k-dimensional groups. In the context of a cube, the data cube core are the base units of aggregation which can be further aggregated to compute other aggregations in the whole cube [6]. For example, given CUBE(A, B, C) with SUM, the groups (a_1, b_i, c_j) for some i can be aggregated to compute SUM((a_1, b_i)), all of which can in turn be aggregated to compute SUM((a_1)). We extend this definition to define the *core* of a set of groups with super and sub-group relationship, which is the basis for bounding.

Definition 1. *(The core groups and the group-Core) Given a group g and the set of its sub-groups $\mathcal{S}_g$, the* group-Core *of g and $\mathcal{S}_g$, denoted as $\mathcal{C}_g$, are the set of sub-groups in $\mathcal{S}_g$ that have the maximum number of grouping-dimensions. Each group in $\mathcal{C}_g$ is a* core group *denoted as g_c.*

All later discussions are implicitly within the context of g and $\mathcal{S}_g$. Also, F^U and F^L denote the upper and lower bound of a distributive or algebraic function F for g and $\mathcal{S}_g$ respectively. The symbols +ve and -ve denote positive and negative.

Like the data cube core, g_c groups are the base units of aggregations, a set of g_c groups can be aggregated to compute g or a group in $\mathcal{S}_g$. Therefore, the largest and smallest aggregations of F that can possibly be produced by any subset of $\mathcal{C}_g$ are the common bounds of F for g and $\mathcal{S}_g$.

While the set of base units can be any set of higher dimensional sub-groups of g_c groups for g and every group in $\mathcal{S}_g$, the g_c groups are the most aggregated base units. Most stringent pruning by bounding is achieved with g_c groups.

It is clear that exhaustively checking all subsets of $\mathcal{C}_g$ is infeasable. Furthermore, there are multiple local aggregate functions in a algebraic function F . We formally define the boundability of an aggregate function as follows.

Definition 2. *An aggregate function F is boundable if (1) the bounds for F are made up from the boundary-values of every local aggregate function f; and (2) enumerating the subsets of $\mathcal{C}_g$ is unnecessary in deriving the boundary-values of every f. The* boundary-values *of f include the max +ve, min +ve, max -ve, and min -ve values of f. The appropriate boundary-value of each f for bounding F is based on the context of the arithmetic operations f is in.*

2.1 Bounding Distributive Functions

All distributive functions are boundable since (1) the boundary-values of the single f prodce the bounds of F and (2) they can be derived with a single scan of $\mathcal{C}_g$. The bounds in terms of the set of g_c groups of all distributive functions for +ve and -ve measures are listed in Table 1.

F	F^U	F^L
SUM	if $\exists$ SUM(g_c) > 0, SUM(SUM(g_c)), where SUM(g_c) > 0; otherwise, MAX(SUM(g_c))	if $\exists$ SUM(g_c) < 0, SUM(SUM(g_c)), where SUM(g_c) < 0; otherwise, MIN(SUM(g_c))
COUNT	SUM(COUNT(g_c))	MIN(COUNT(g_c))
MAX	MAX(MAX(g_c))	MIN(MAX(g_c))
MIN	MAX(MIN(g_c))	MIN(MIN(g_c))

Table 1. The upper and lower bounds of all distributive functions

2.2 Bounding Algebraic Functions

Bounding covers algebraic functions that apply only the basic arithmetic operations (+, −, ×, and /) on their local aggregate functions.

The boundability of algebraic functions is illustrated by the following two examples. Assume the measure can be +ve or -ve and let s and c be the local aggregate values *sum* and *count* at g_c groups.

Example 1. (A boundable algebraic function) The algebraic function AVG = SUM(s)/SUM(c). The max +ve of SUM(s) and the min +ve of SUM(c) derive AVG^U. Also both the max +ve SUM(s) and the min +ve SUM(c) are boundable, as is shown in Table 1. AVG is boundable.

Example 2. (An unboundable algebraic function) The algebraic function (1/AVG) = SUM(c)/SUM(s). The max +ve of SUM(c) and the min +ve of SUM(s) derive (1/AVG)^U. Unfortunately, the min +ve of SUM(s) comes from some combinations of g_c groups which cannot be determined by a single scan of $\mathcal{C}_g$. (1/AVG) is unboundable.

2.3 Optimisation for the expression "SUM(a) /SUM($+b$)"

We optimise the common expression "SUM(a) / SUM($+b$)" which exists in many algebraic functions, including AVG, CenterOfMass, and VAR (SD). For example, VAR = SUM(s^2)/SUM(c) − 2 × (SUM(s)/SUM(c))^2 + SUM(s)/SUM(c) where s^2, s, and c are the square of sum, sum and count values at g_c groups and SUM(c) is always positive. The function contains three expressions of the form SUM(a)/SUM($+b$).

Theorem 1. *(Bounding the expression E = SUM(a)/SUM($+b$)) Given E, let a_i and b_i be two intermediate aggregate values a and b in the i^{th} g_c group and b_i is always positive. Then E^U = MAX(a_i/b_i) and E^L = MIN(a_i/b_i).*

Due to space constraints, the formal proof is omitted. The optimised E^U and E^L are more stringent than the general techniques presented in Section 2.2. We use AVG to liiustrate that the optimisation achieves more strigent bounding. AVG^U is SUM(s)^U/SUM(c)^L with the techniques in Section 2.2, and it is MAX(a/b) with the optimisation. Since SUM(a)$^U \geq a_i$ and SUM(b)$^L \leq b_i$ for any a_i and b_i, SUM(a)^U/SUM(b)$^L \geq$ MAX(a/b).

In the next section, we present an efficient aggregation strategy which can easily identify $\mathcal{C}_g$ for incorporating bounding during computation.

3 Top-Down Multiway Aggregation

We adopt a strategy where multiple cuboids are aggregated simultaneously. The *Group-Bounding Tree* (*GB-tree*) is built to directly represent multiple cuboids. The simultaneous computation of multiple remaining cuboids in a data cube involves the construction of sub-GB-trees. Bounding with $\mathcal{C}_g$ are incorporated for pruning branches while building a sub-GB-tree.

3.1 The Group-Bounding Tree

A Group-Bounding Tree (*GB-tree*) of an aggregation F for a k-dimensional dataset consists of a root node and k levels of branch nodes, each of which corresponds to a dimension. A node contains a dimension-value, one or more intermediate aggregate values for F, and the links to its child nodes. An example GB-tree on dimensions A, B, C, D, and E for AVG is presented in Figure 2.

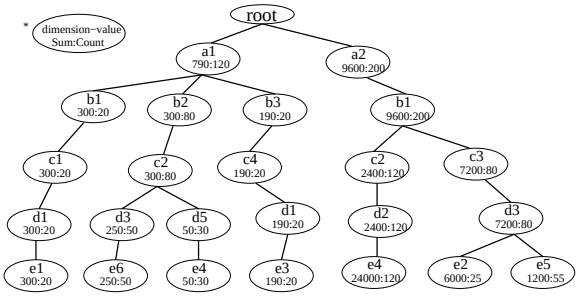

Fig. 2. *Group-Bounding Tree*

Groups and sub-groups in the GB-tree In a GB-tree, a group g is represented by the nodes of g's last grouping dimension-values on the branches containing all of g's grouping dimension-values. A sub-group of g is represented on a subset of g's branches that contain the additional grouping-dimension-values of the sub-group. In the example, `(a1,b1)` is a sub-group of `(a1)` but not of `(b1)`. Two observations are made:

Observation 1 *On a GB-Tree, the branch from the root to a node is a unique combination of dimension-values. Each node represents a unique group and a descendent node its sub-group.*

Observation 2 *The leaf nodes under a group g are g's sub-groups having the largest number of grouping-dimensions. They are the g_c groups of g and $\mathcal{S}_g$.*

3.2 The Multiway Aggregation Strategy

Building a GB-tree simultaneously computes many groups. In the example, groups in cuboids ALL, A, AB, ABC, $ABCD$, $ABCDE$ are computed. For the remaining cuboids, each non-leaf dimension of the GB-tree is dropped to construct a sub-GB-tree; and each non-leaf dimension that is below the last dropped dimension of a sub-GB-tree is subsequently dropped to further construct sub-GB-trees. When dropping a dimension D on a GB-tree, the branches below each d_i ($d_i \in D$) are amalgamated to form the new set of descendent levels of the parent of the dropped dimension.

Each sub-GB-tree computes the set of child cuboids at the levels below the last dropped dimension; they share the prefix dimensions before the last dropped dimensions. Let a group grouped by the prefix dimensions be a *prefix group*; groups subsequently aggregated are all its sub-groups.

In our example, the sub-trees of the original tree are $BCDE$ ($-A$), $ACDE$ ($-B$), $ABDE$ ($-C$), and $ABCE$ ($-D$) where the last dropped dimension is indicated in (). The tree $ACDE$, whose last dropped dimension is B and the prefix dimension is A, computes the cuboids $ACDE$, ACD, and AC; they are all sub-groups of some a_i. The cuboids ADE and AD computed by further dropping C of $ACDE$ sub-tree are also sub-groups of some a_i.

The recursive construction of sub-trees by dimension-dropping computes all cuboids. The Bound-cubing Algorithm is shown in Figure 3.

```
Global Input: GB-tree T of a k-dimensional dataset, c =constraint.
              //T.prefixCuboid= ∅ and T.last-droppedD = 0
Output: Bound-cubing(T)
Algorithm Bound-cubing(T) {
    foreach (g ∈ T)// g is a group on a node of T
        if(constraintCheckingOK(g)) output g;
    subTrees = T.dropDimensions();
    foreach (sT ∈ subTrees) Bound-cubing(sT);
}
Procedure dropDimensions() {
    subTrees = ∅;
    foreach (i ∈ {last-droppedD +1, ..., k} )
        create sub-tree sT_i;
        foreach (g_p ∈ prefixCuboid)// foreach prefix group
            if(boundCheckingOK(g_p))
                sT_i.amalgamate(g_p, D_i);//amalgamate branches of g_p below D_i
        sT_i.last-droppedD= i;
        sT_i.prefixCuboid= prefixCuboid ∪ {∀ D_j | j ∈ {1 ... i−1}}
        subTrees = subTrees ∪ sT_i;
    return subTrees;
}
```

Fig. 3. *Bound-Cubing Algorithm*

The computation strategy diagram for CUBE(A, B, C, D, E) is shown in Figure 4. The cuboids in each rectangle are computed simultaneously from a single GB-tree.

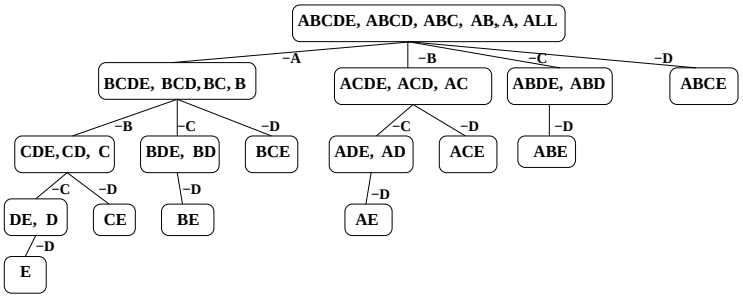

Fig. 4. *The Aggregation Strategy for all cuboids*

Stringent Bounding for Pruning A sub-GB-tree computes sub-groups of some prefix group g_p. The leaf nodes of g_p are the g_c groups of g_p and its sub-groups. Pruning using g_c groups of g_p can be directly applied when building a sub-GB-tree. The branches under failed g_p are trimmed so that they do not participate in generating sub-trees. To ensure effective pruning, when constructing the next level sub-trees, the prefix groups are updated and the new g_c groups are used for pruning. To illustrate, the prefix groups on $ACDE$ tree is A; by dropping D, ACE tree is formed which computes the cuboid ACE. The prefix

groups become AC and the ACE groups at the leaf level become the new g_c groups.

4 Related Work

We discuss the two most related work, the Divide-and-Approximate [9] and Star-cubing [4] iceberg cubing algorithms.

Divide and Approximate-cubing (D&A pruning) While bounding can handle some algebraic functions with local aggregations of both signs in $\times$ and / operations, they cannot be handled by D&A [9]. The constraint " SUM/MAX" cannot be approximated by D&A pruning when SUM and MAX can be both positive and negative values, but they can be bounded. Moreover, in D&A pruning, the approximator can only be derived from some group g with a sub-group of g, g_s, that is grouped by all dimensions and can prune only the groups that are sub-groups of g and super-groups of g_s. This suggests that in the simultaneous computation of multiple cuboids, it is difficult to locate the specific groups that can be pruned by the derived approximator on the tree before the groups are actually computed.

Star-cubing Both the Star-tree and the GB-tree are inspired by the H-tree [7]. Star-cubing uses star-nodes which represent all non-frequent dimension-values to reduce the size of the Star-tree. It is shown in [4] that for monotone constraints, star nodes significantly improve the efficiency of iceberg cubing. It is briefly mentioned that star nodes can also be incorporated for non-monotone constraints such as those involving AVG. However, this may involve considerable additional cost. Star nodes can be easily incorporated in Bound-cubing for monotone constraints.

Bound-cubing and Star-cubing share the spirit of multiway computation of cuboids [10]. In Star-cubing, with a traversal of the Star-tree, only the group at the root and the groups of the cuboid at the leaf level are computed. Bound-cubing computes multiple cuboids at multiple levels of the GB-tree at once while multiple Star-trees need to be built in Star-cubing.

5 Experiments

Two datasets are used for our experiments. One is real world data consisting of 88443 tuples taken from the 1990 US census [12]. The dataset has 61 attributes. We extracted the 10 attributes with discrete domains as dimensions such as group-quarters-type, marital-status, and occupation, with cardinalities between 6 to 10. The measure is total-income. The other is the OLAP benchmark relational tables *TPC-R* [1]. We constructed a single joined relation to derive a multidimensional dataset consisting of 1 million tuples with 10 dimensions, with cardinalities between 2 and 25.

We examine two aspects of Bound-cubing separately. First, the aggregation strategy of Bound-cubing is compared with Star-cubing. Second, the bounding

techniques are compared with D&A for pruning effectiveness. For fair comparison, we have implemented all three algorithms and applied the same implementation optimisations whenever applicable. Our experiments were conducted on a PC with an Intel Pentium R 4, 1.70 GHz CPU and 512M main memory, running Red Hat Linux7.2. All programs are written in C++. All data structures required can fit in memory. The runtime measured excludes the I/O time.

5.1 Bound-cubing vs. Star-cubing

To evaluate the aggregation cost of Bound-cubing, we compare it with Star-cubing using the monotone constraint "$\mathsf{COUNT}(*) \geq \delta$". The runtime of both algorithms on both datasets are recorded at different count thresholds and shown in Figure 5. Bound-cubing is consistently faster than Star-cubing on both datasets at all count thresholds. Bound-cubing is slightly more efficient than Star-cubing. This can be attributed to the more simultaneous aggregation in Bound-cubing.

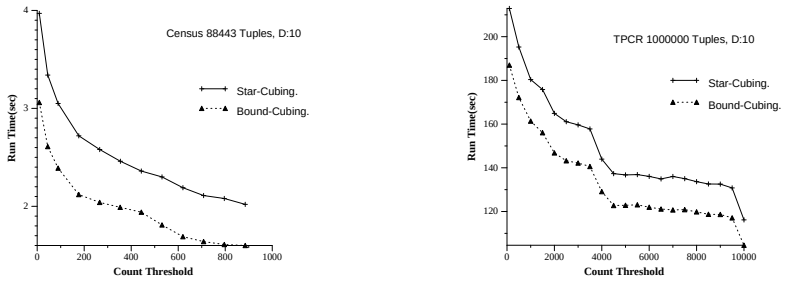

Fig. 5. Bound-cubing vs Star-cubing for Count Threshold

5.2 Bound-cubing vs. D&A cubing

We compare Bound-cubing with D&A cubing on the non-monotone constraint "$\mathsf{AVG}(m) \geq \alpha$". Their performance is shown in Figure 6 Two observations are made: (1) Bound-cubing is always significantly more efficient than D&A cubing at all thresholds. The improvement is 2 to 15 times on census dataset and 7 times on TPCR dataset. The performance gain is attributed to the simultaneous aggregation and bounding for pruning. (2) While the threshold of the constraint increases, Bound-cubing becomes faster. Surprisingly, the runtime of D&A does not decrease with larger thresholds. This suggests that the overhead of pruning increases with the increase in the constraint threshold. In contrast, bounding techniques have low overhead and the efficiency gains always outweigh the overhead incurred in pruning.

6 Conclusions and Future work

We have developed effective techniques that bound the aggregate values of groups. The bounding techniques are general and can prune for complex non-monotone constraints defined with distributive and algebraic functions. We have

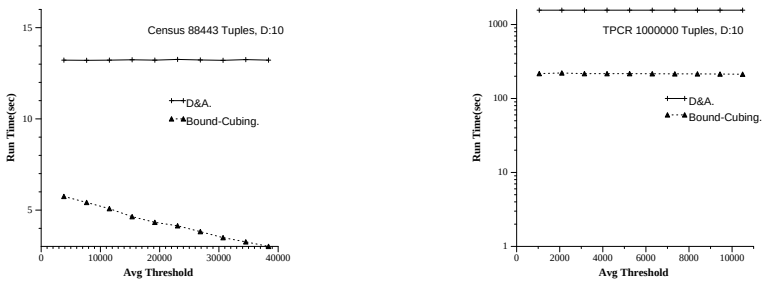

Fig. 6. Bound-cubing vs D&A for AVG Threshold

also developed an efficient computation strategy on the Group-Bound tree that computes multiple cuboids simultaneously. In terms of cube computation, our contribution is a general approach for dealing with complex non-monotone constraints. The approach incurs little overhead and fits nicely with the multiway aggregation strategy.

7 Acknowledgements

We thank Justin Zobel for his helpful comments.

References

1. The TPC-R benchmark, http://www.tpc.org/tpcr/.
2. Beyer and Ramakrishnan. Bottom-up computation of sparse and Iceberg CUBE. SIGMOD'99.
3. Agarwal et al. On the computation of multidimensional aggregates. VLDB'96.
4. Dong et al. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. VLDB'03.
5. Fang et al. Computing iceberg queries efficiently. VLDB'98.
6. Gray et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. J. Data Mining and Knowledge Discovery, 1997.
7. Han et al. Efficient computation of iceberg cubes with complex measures. SIGMOD'01.
8. Sarawagi et al. On computing the data cube. Tech. report, IBM Almaden Research Center, 1996.
9. Wang et al. Pushing aggregate constraints by divide-and-approximate. ICDE'02.
10. Zhao et al. An array-based algorithm for simultaneous multidimensional aggregates. SIGMOD'97.
11. Ross and Srivastava. Fast computation of sparse datacubes. VLDB'97.
12. Historical Census Projects. ftp://ftp.ipums.org/ipums/data/ip19001.Z, 1997.