Computing Iceberg Quotient Cubes with Bounding

Xiuzhen Zhang[1], Pauline Lienhua Chou[1], and Kotagiri Ramamohanarao[2]

[1] School of CS & IT, RMIT University, Australia
{zhang, lchou}@cs.rmit.edu.au
[2] Department of CSSE, The University of Melbourne, Australia
rao@csse.unimelb.edu.au

Abstract. In complex data warehouse applications, high dimensional data cubes can become very big. The quotient cube is attractive in that it not only summarizes the original cube but also it keeps the roll-up and drill-down semantics between cube cells. In this paper we study the problem of semantic summarization of iceberg cubes, which comprises only cells that satisfy given aggregation constraints. We propose a novel technique for identifying groups of cells based on *bounding* aggregates and an efficient algorithm for computing iceberg quotient cubes for monotone functions. Our experiments show that iceberg quotient cubes can reduce data cube sizes and our iceberg quotient cubing algorithm can be over 10-fold more efficient than the current approach.

1 Introduction

Since the introduction of the Cube operator [3], interests on data cube research has grown substantially. Several cube computation algorithms have been proposed, including relational approaches PipeHash and PipeSort [1], MemoryCube [7] and multiway array aggregation [13]. The number of cube cells grows exponentially with the number of dimensions. Large data cubes are difficult for storage and answering queries. Recent studies have focused on how to compute compressed data cubes, including Condensed Cube [9], Dwarf [8], Quotient Cube [5] and QC-Tree [6]. The quotient cube is especially attractive in that it compresses the original cube as well as keeps the roll-up/drill-down semantic among cells.

Iceberg cubes comprise cube cells whose aggregate value satisfies a given constraint. Many algorithms for iceberg cube computation have been proposed, including BUC [2], H-Cubing [4] and Star-Cubing [11]. Since cells failing the aggregation constraint are removed from the solution, there are "holes" in the lattice structure for iceberg cubes (shown in Fig. 1). It is interesting to see, with the presence of such "holes" of removed cells, whether semantic summarization can compress the original iceberg cubes. It is also interesting to study if iceberg cubes can be efficiently summarized while keeping the semantics. A tricky problem is how to incorporate effective pruning into summarization.

A Min Tjoa and J. Trujillo (Eds.): DaWaK 2006, LNCS 4081, pp. 145–154, 2006.

In this paper, we propose the concept of iceberg quotient cube and study its efficient computation. An iceberg quotient cube comprises classes of cells that satisfy a given constraint, and in each partition cells are of equal aggregate values and are connected by the roll-up/drill-down relationship. Obviously computing the iceberg cube first and then summarizing the resulting cells is a time-consuming approach. It is more efficient the iceberg quotient cubes are computed from base tables and pruning is applied with aggregation.

1.1 Main Ideas

We apply a novel technique *bounding* [12] for identifying lattices (of cells) of equal aggregates while pruning unqualified lattices.

In Table 1(a) Month, Product, SalesMan and City are dimensions, and Sale is the measure. Similarly Table 1(b) is a 4-dimensional dataset. A data cube of 4 dimensions comprises the 16 group-bys (including the empty group-by) from any subset of the 4 dimensions. With an aggregate function, each group-by in a data cube generates aggregations of the multi-set of measure values for partitions of tuples with the equal dimension-values, which we call *cells*. For example in Table 1(a) Min and Count are aggregate functions, and $(\texttt{Jan}, \texttt{Toy}, \texttt{John}, \texttt{Perth})$ is a cell with aggregations of $\text{Min}(\texttt{Sale}) = \texttt{200}$ and $\text{Count}(*) = \texttt{5}$.

A data cube is a lattice with top and bottom cells respectively. The lattice on the left of Fig. 1 is the cube lattice for the toy dataset in Table 1(b). Cells are related by the super-cell/sub-cell relationship. Following the convention of BUC [2], a sub-cell (with more dimensions) are above its super-cells. The top cell for the lattice is False (not shown in Fig. 1), the empty cell that does not aggregate any tuples. The bottom cell is (*, *, *, *), aggregating all tuples (* matches any value). The bounds for a lattice are computed from the most specific cells (MSCs) in the lattice under consideration. The MSCs can be viewed as the basic units for computing data cubes as all other cells can be computed from the MSCs. Table 1(a) shows a 4-dimensional dataset with 6 MSCs. $\text{Min}(\texttt{Sale})$ decreases monotonically with super-cells. The lower bound for the data cube is the minimum of $\text{Min}(\texttt{Sale})$ for all MSCs, which is 100. The upper bound for the cube is the maximum among all MSCs, which is 200. As will be seen later, a data cube can be decomposed into a set of sub-cubes and the bounding from MSCs applies to sub-cubes as well.

The 3 MSCs with "Month=March" form a sub-lattice, with $(\texttt{Mar}, *, *, *)$ at the bottom and False at the top. What is special about this lattice is that its upper and lower bounds are both 100 (bounds are calculated as described before). So the lattice represents a class of cells with $\text{Min}(\texttt{Sale}) = \texttt{100}$. If the aggregate value satisfies a given constraint, then it becomes a temporary class in the solution; otherwise the class is pruned. For monotone aggregate functions, such temporary classes are efficiently merged to produce the maximal partition of an iceberg data cube.

Table 1. Two sample dimensional datasets

Month	Product	SalesMan	City	Min(Sale)	Count(*)
Jan	Toy	John	Perth	200	5
Mar	TV	Peter	Perth	100	40
Mar	TV	John	Perth	100	20
Mar	TV	John	Sydney	100	10
Apr	TV	Peter	Perth	100	8
Apr	Toy	Peter	Sydney	100	5

(a) A sales dataset, partially aggregated

A	B	C	D	Sale
a_1	b_1	c_1	d_1	650
a_1	b_1	c_2	d_1	322
a_1	b_1	c_2	d_1	1087

(b) A toy dataset

1.2 Related Work

Our concept of iceberg quotient cube is motivated by the quotient cube [5]. We introduce semantic summarization into iceberg cubes. More importantly our approach of bound-based pruning and computing of iceberg quotient classes is different from the previous tuple-based approach [5]. A "jumping" method that can identify an equivalence class of cells without examining all cells in the class is essential for the efficiency of quotient cube computation. Based on BUC [2], Lakshmanan et al. [5] proposed a jumping method that involves examining every record in a partition of the underlying dataset. In contrast, our bound-based jumping method identifies an equivalence class of cells by examining MSCs; such an approach can improve the efficiency of quotient cube computation.

The QC-Tree is a data structure for storing quotient cubes. It is orthogonal to and can complement our work on iceberg quotient cubes.

The Dwarf Cube [8] and Iceberg Dwarf Cube [10] compresses the cube cells by exploiting shared prefixes and suffixes. The Condensed Cube [9] compresses a data cube by condensing the cells aggregated from the same set of base relation tuples into one cell. Nevertheless the focus of all these work are on compression and the semantics of data cubes are lost in the process.

Bound-prune cubing was proposed in our previous work to compute iceberg cubes [12]. In this work we apply bounding to summarization of data cubes.

2 Iceberg Quotient Cubes

The important roll-up and drill-down semantics on a data cube is the super-cell/sub-cell relationship among cells. Lakshmanan et al. [5] proposed the basic definitions for quotient cubes that preserve such semantics. Generally a data cube is partitioned into convex classes of cells with equal aggregates and cells in a class have the super-cell/sub-cell relationship.

Definition 1 Convex connected equivalence class. *All cells in a connected equivalence class are related by the super-cell(sub-cell) relationship and have equal aggregate values. In a convex class P, if a cell g and a sub-cell g' are in P, then cells that are sub-cells of g and super-cells of g' are also in P.*

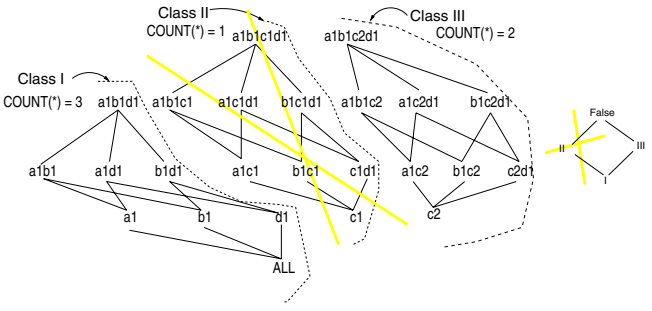


Fig. 1. The quotient cube of the function Count(∗) for the dataset in Table 1(b)

Definition 2 Quotient Cube Lattice. *A quotient cube lattice consists of convex connected equivalence classes of cells. Classes in the quotient lattice are connected by the super-class/sub-class relationship: a class C is super-class (sub-class) of another class D if there exist cells $c \in C$ and $d \in D$ such that c is a super-cell (sub-cell) of d. Each equivalence class is denoted as $[B, T]$, where B and T are the set of cells at the bottom and top of the class respectively.*

Example 1. *Consider the 4-dimensional dataset in Table 1(b). The original cube-lattice for* Count(∗) *has 24 cells, shown in Fig. 1. Cells can be summarized into 3 classes, which are represented by the bottom and top cells. Class I is* $[\{(\mathsf{ALL})\}, \{(a_1, b_1, d_1)\}]$. $[\{(c_1)\}, \{(a_1, b_1, c_1, d_1)\}]$ *and* $[\{(c_2)\}, \{(a_1, b_1, c_2, d_1)\}]$ *are Class II and Class III respectively. Class II is a sub-class of class I. all cells in class II are sub-cells of some cells in Class I but not super-cells.*

Definition 3 Optimal Quotient Cube Lattice. *A quotient cube is optimal if all of its classes are* maximal. *A class is maximal if it contains the largest set of cells with equal aggregate values while satisfying connectivity and convexity.*

The aggregate value of a monotone aggregate function increases or decreases monotonically with respect to the super-cell relationship. For example, Count(∗) values increase with super-cells, whereas Min values decrease with super-cells. For monotone functions, there is a unique optimal quotient cube partition that coincides with the partition induced by connected equivalence partitions. In other words, if all connected cells with equal aggregate values are clustered in one class, the result is the optimal quotient cube. For non-monotone functions, a connected equivalence class is not necessarily convex, therefore, the optimal quotient cube cannot be induced solely from connected equivalence partitions.

Having all the basic definitions from [5], we are now ready to introduce our new definitions. The observation below emphasizes the following fact: For a given constraint, given that all cells in a class have the same aggregate value, a class is either pruned entirely or remains as a class, in other words, it is never split as the result of pruning by the aggregation constraint.

Observation 1. *Given a constraint, all cells failing the constraint form classes in the quotient cube. An iceberg cube consists of cells in the classes of the quotient cube whose aggregate values pass the constraint. The roll-up and drill-down semantics among the qualified classes are preserved.*

Definition 4 Iceberg Quotient Cube. *Let cells satisfying a given constraint be called iceberg cells. All iceberg cells are partitioned into convex and connected equivalence classes. The classes form an iceberg quotient cube.*

Example 2. *Continuing with Example 1, consider the constraint "*$\text{Count}(*) \geq 2$*". Class I and III in Fig. 1 remain while the entire Class II is pruned, as is denoted by the cross in Fig. 1. The semantics between Class I and III is kept.*

The crucial question to answer now is how to identify the equivalence classes of cells while effectively prune unpromising cells to achieve efficient iceberg quotient cubing. Our novel *Bounding* technique can solve both questions.

3 Computing Iceberg Quotient Cubes with Bounding

The naive approach of computing an iceberg cube first and then summarizing it into a quotient is obviously not an efficient approach. A more efficient approach of computing iceberg quotients is to compute iceberg quotients directly from input datasets, where aggregation, pruning with constraints and summarization is performed at the same time. Bounding can efficiently identify equivalence classes in a cube lattice with little extra cost. For monotone aggregate functions, the classes can then be easily merged to produce a set of maximal equivalence classes. We also present an efficient iceberg quotient cubing algorithm that incorporate all these ideas.

3.1 Bounding Aggregate Functions

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 MSCs of the data cube. However exhaustively checking the power set of MSCs is not computationally feasible. An aggregate function F is *boundable* [12] for a data cube if some upper and lower bounds of F can be determined by an algorithm with a single scan of some auxiliary aggregate values of MSCs of the data cube. We use an example to explain the main ideas of bounding. Details are described in [12].

Example 3. *Given measure* X, $\text{Count}(X) = \text{Sum}(\{\text{Count}(X_i) \mid i = 1..n\})$, *where* X_1, ..., X_n *are MSCs. The number of tuples in a cell of a data cube is no larger than the total number of tuples of all MSCs. Suppose* g *is a cube cell,* $\text{Count}(g) \leq \text{Sum}(\{\text{Count}(X_i) \mid i = 1..n\})$. *So* $\text{Sum}(\{\text{Count}(X_i) \mid i = 1..n\})$ *is an upper for Count(X) of the data cube. On the other hand, to compute the lower bound, we also have* $\text{Count}(g) \geq \text{Min}(\{\text{Count}(X_i) \mid i = 1..n\})$. *As a result, the lower bound is* $\text{Min}(\{\text{Count}(X_i) \mid i = 1..n\})$. *Both bounds can be obtained by one scan of MSCs and Count is boundable.*

Table 2. The bounds of SQL aggregate functions

F	upper bound ; lower bound
Count	$\underset{i}{\text{Sum}}\ \text{Count}(X_i)$; $\underset{i}{\text{Min}}\ \text{Count}(X_i)$
Max	$\underset{i}{\text{Max}}\ \text{Max}(X_i)$; $\underset{i}{\text{Min}}\ \text{Max}(X_i)$
Min	$\underset{i}{\text{Max}}\ \text{Min}(X_i)$; $\underset{i}{\text{Min}}\ \text{Min}(X_i)$
Sum	$\underset{\text{Sum}(X_i)>0}{\text{Sum}}\ \text{Sum}(X_i)$ if (a) $\underset{i}{\text{Max}}\ \text{Sum}(X_i)$ otherwise ; $\underset{\text{Sum}(X_i)<0}{\text{Sum}}\ \text{Sum}(X_i)$ if (b) $\underset{i}{\text{Min}}\ \text{Sum}(X_i)$ otherwise
Average	$\underset{i}{\text{Max}}\ \text{Avg}(X_i)$; $\underset{i}{\text{Min}}\ \text{Avg}(X_i)$

Given a dataset, X is the measure and X_1, ..., X_n are the MSCs.
(a): there is i such that $\text{Sum}(X_i) > 0$. (b): there is i such that $\text{Sum}(X_i) < 0$.

All SQL aggregate functions Count, Min, Max, Sum and Average are boundable, and their bounding algorithms are listed in Table 2. Note that Count, Max, Min, and Sum on non-negative (non-positive) measure values are monotone functions whereas Sum on arbitrary values and Average are non-monotone functions.

3.2 Identifying Equivalence Classes with Bounding

Observe that lattices are connected and convex. Bounding can be used to detect if a cell-lattice is an equivalence class as stated in the following proposition.

Proposition 1. *Given a lattice, all cells in the lattice have equal aggregate values if the upper and lower bounds of the lattice are equal.*

Proof. Proof of the proposition follows directly from that the aggregate values of cells in a lattice are bounded by the lower and upper bounds.

A data cube lattice can be partitioned into a set of sub-lattices each of which has equal aggregates. For all monotone functions, such a partition is easily achieved with single depth-first traversal of the G-tree (Section 3.3). If the bounds of a sub-lattice are equal and pass the constraint, it can be identified as one single class. The cells in the lattice do not need to be computed.

Although the temporary equivalence classes on a lattice is maximal with respect to a sub-lattice, to obtain the global maximal equivalence classes the temporary equivalence classes should be merged. Following Theorem 2 of [5], the following remark for monotone functions allows the merging process to produce maximal connected and convex equivalence classes based on the connectivity of classes and equality of their aggregate values.

Remark 1. For monotone functions, the unique optimal iceberg quotient cube is the partition induced by the connected equivalence classes.

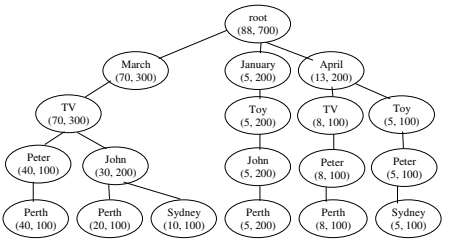


Fig. 2. The first G-tree Table 1(a) cube

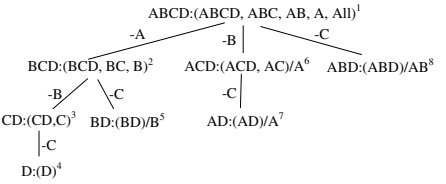


Fig. 3. The G-trees for Cube(ABCD)

The following observation states how merging is achieved. The main idea is to derive more general and specific cells respectively from the bottom and top cells of lattices to be merged.

Observation 2. *For monotone functions, if two convex equivalence classes C_1 and C_2 are connected, the two classes can be merged into a coarser class C as follows: The bottom cells of C are the minimal set for the bottom cells of C_1 and C_2 with respect to the sub-cell relationship. The top cells of C are the maximal set for the top cells of C_1 and C_2 with respect to the sub-cell relationship.*

From Observation 2, to compute an iceberg quotient cube for a monotone function, two classes of equal aggregate values are merged as long as they have the super-cell/sub-cell relationship.

Example 4. *For simplicity * is omitted in cell notations. Consider merging*
$C_1 = [\{(a_1, b_1, c_1)\}, \{(a_1, b_1, c_1, d_1)\}]$
and
$C_2 = [\{(b_1, c_1), (b_1, d_1)\}, \{(a_1, b_1, c_1), (a_1, b_1, d_1)\}]$.
The resulting class C is $\langle\{(b_1, c_1), (b_1, d_1)\}, \{(a_1, b_1, c_1, d_1)\}\rangle$. Among the bottom cells of C_1 and C_2, $\{(a_1, b_1, c_1), (b_1, c_1), (b_1, d_1)\}$, (b_1, c_1) is a super-cell of (a_1, b_1, c_1). So the minimal set is $\{(b_1, c_1), (b_1, d_1)\}$ and becomes the bottom cells of C. The top cells of C are similarly derived.

3.3 The G-Tree

The data structure for computing quotient cubes is the *G-tree* [12]. We use the first G-tree for cubing the dataset in Table 1(a) as an example to explain, which is shown in Fig. 2. In each node are the aggregates Sum(`Sale`) and Count(∗). The aggregates in each node are for the cell with dimension-values on the path from the root to the node. For the leftmost path from the root of the G-tree, in the node (`Peter`) there are 40 tuples with Sum(`Sale`) = 100 in the (`March`, `TV`, `Peter`, ∗) partition. The leaf nodes give the MSCs for Cube (`Month`, `Product`, `SalesMan`, `City`).

To compute cells not represented on the first G-tree, sub-G-trees [3] are recursively constructed by collapsing dimensions. Fig. 3 shows the G-trees for

[3] Note that a sub-G-tree is not part of the original G-tree, but obtained from the original G-tree by collapsing a dimension.

Input: a) An N-dimensional dataset D with measure m.
b) Aggregation constraint $C(F)$, where the aggregate function F is monotone.
Output: the Iceberg quotient cube Q, assumed global.
(1) Build the G-tree T from D for F.
(2) $Q = \phi$;
(3) **BIQC**(T, F, C);

Procedure BIQC (T, F, C)
(1) Let g be a conditional cell on T and L_g denote g's lattice of cells
(2) Compute the bounds $[B_1, B_2]$ for L_g;
(3) **if** (both B_1, B_2 violate C)
(4) Skip the processing of L_g;
(5) **else if** ($B_1 == B_2$)
(6) Merge the class from L_g to Q; //skip the processing of L_g, Section 3.2.
(7) **else**
(8) $S_L \leftarrow$ sub-lattices with equal bounds by depth-first traversal;// Section 3.1.
(9) Merge classes in S_L to Q;
(10) **for each** dropping dimension D on T **do**
(11) $T_s \leftarrow$ the sub G-tree from collapsing D from T; //pruning, Section 3.4
(12) **BIQC**(T_s, F, C);

Fig. 4. The Bound Iceberg Quotient Cubing Algorithm

Cube(ABCD). Each node represents a G-tree. For the ABCD-tree at the top, the corresponding group-bys are (A,B,C,D), (A,B,C), (A,B), (A) and (). The sub-G-trees of the ABCD-tree, which are the (−A)BCD, A(−B)CD, and AB(−C)D trees, are formed by collapsing on dimensions A, B, and C respectively. The dimensions after "/" in each node denote common prefix dimensions for the tree at the node and all its sub-trees. In the sub-G-tree construction process, we compute the bounds based on prefix dimensions and use them for pruning [12].

3.4 The Bound Iceberg Quotient Cubing Algorithm

Our bound iceberg quotient cubing (BIQC) algorithm is shown in Fig. 4. Line 1 of the algorithm denotes the following process of identifying temporary equivalence lattices: Following the depth-first traversal of T, at the node of a prefix cell g, a class C is formed and g is both the top and bottom cell of the class. If a descendent node g_d of g has equal aggregate value to that of g, then g_d is added as a top cell for G. Any sub-cell of g_d as a top cell is replaced by g_d. For any descendent cell $g_{d'}$, if the aggregate value is different from that of C, a new class $C^{'}$ with $g_{d'}$ as the bottom cell is created. On returning from the recursion to the bottom-cell of a class, the class is completed.

Merging classes is at lines 6 and 9. When a lattice under consideration is not an equivalence class, it is partitioned into equivalence classes(Section 3.2). Pruning is applied at line 4 and line 11 on lattices whose bounds fail the constraints. Especially at line 11, all cells yet to be computed from the branches already included in an equivalence class or failing the constraint are pruned.

4 Experiments

We did experiments to study the compression effectiveness of iceberg quotient cubes and the efficiency of BIQC for the constraint "Count(∗) ≥ α". We compare BIQC with BUC, the underlying iceberg cubing algorithm in [5]. To accurately compare computation cost, timing does not include the time for writing output.

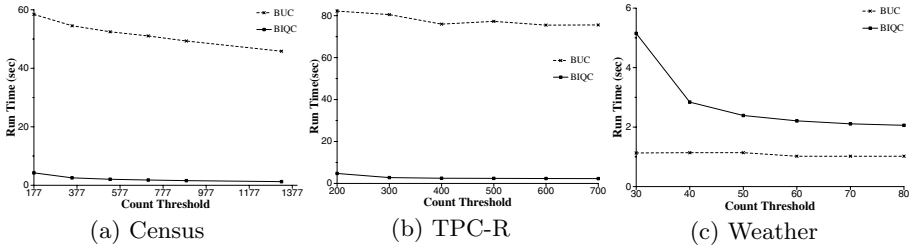
Three datasets are used in our experiments. The US Census dataset [4] is dense and skewed: 88,443 tuples, 12 dimensions, and a cardinality range of 7–48. The TPC-R[5] dataset is relatively dense and random: 1000,000 tuples, 10 dimensions, and a cardinality range of 3–25. The Weather dataset [6] is extremely sparse: 100,000 tuples, 9 dimensions, and a cardinality range of 2–6505.

4.1 Compression Effectiveness of Iceberg Quotient Cubes

The compression ratio is the number of classes in a iceberg quotient as the proportion of number of cells in the original iceberg cube. A quotient cube with lower compression ratio is more effective in compressing the original data cube. With the constraint "support(*) ≥ α" (support is the relative Count in percentage), the compression ratio for the three datasets remains almost constant for $\alpha = 10\%..80\%$. On the sparse weather dataset, the compression ratio is around 80%. In contrast on the dense and skewed census dataset it is around 20%. This low ratio is in contrast to that of 70% on the random TPC-R dataset, It can be seen that iceberg quotient cube can more effectively reduce iceberg cube size on dense and skewed data.

4.2 Efficiency of Bound Iceberg Quotient Cubing

Fig. 5 shows the runtime of BIQC on computing the iceberg quotient cube with constraint "Count(*) ≥ α" in comparison to BUC on computing the original iceberg cube. BIQC is over 10-fold more efficient than BUC on Census and



(a) Census (b) TPC-R (c) Weather

Fig. 5. Runtime comparison: BIQC vs. BUC with "Count(*) ≥ α"

[4] ftp://ftp.ipums.org/ipums/data/ip19001.Z.

[5] http://www.tpc.org/tpcr/.

[6] http://cdiac.ornl.gov/ftp/ndp026b/SEP85L.DAT.Z.

TPC-R for all Count thresholds. It is slower than BUC on the extremely sparse Weather data. The likely reason can be that on sparse data there are much more temporary equivalence classes while their sizes are smaller, which increase the cost for merging. BIQC scales relatively well with lower Count thresholds.

5 Conclusions

We have proposed the iceberg quotient cube for semantic summarization of iceberg cubes. We apply a novel technique bounding for efficient computation. Our experiments demonstrated that iceberg quotient cubes are effective for compressing iceberg cubes and our algorithm is significantly more efficient than the existing approach. Our future work will focus on the open problem of computing iceberg quotient cubes for complex aggregate functions [5].

References

1. S. Agarwal et. al. On the computation of multidimensional aggregates. In *Proc. VLDB*, 1996.
2. K Beyer and R Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. SIGMOD*, 1999.
3. 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.
4. J Han et al. Efficient computation of iceberg cubes with complex measures. In *Proc. SIGMOD*, 2001.
5. L. Lakshmanan et al. Quotient cube: How to summarize the semantics of a data cube. In *Proc. VLDB*, 2002.
6. L. Lakshmanan et al. QC-trees: An efficient summary structure for semantic OLAP. In *Proc. SIGMOD*, 2003.
7. K. A. Ross and D. Srivastava. Fast computation of sparse data cubes. In *Proc. SIGMOD*, 1997.
8. Y. Sismanis et. al. Dwarf: Shrinking the petacube. In *Proc. SIGMOD*, 2002.
9. W. Wang et al. Condensed cube: An effective approach to reducing data cube size. In *Proc. ICDE*, 2002.
10. L. Xiang and Y. Feng. Fast computation of iceberg dwarf. In *Proc. SSDBM*, 2004.
11. D. Xin et al. Star-cubing: computing iceberg cubes by top-down and bottom-up integration. In *Proc. VLDB*, 2003.
12. X. Zhang, L. Chou and G. Dong. Efficient computation of iceberg cubes by bounding aggregate functions. *IEEE TKDE*, 2006. In submission.
13. Y. Zhao et al. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc SIGMOD*, 1997.