Computing Full and Iceberg Datacubes Using Partitions

(Extented Abstract)

Marc Laporte[1], Noël Novelli[2], Rosine Cicchetti[1,3], and Lotfi Lakhal[1,3]

[1] IUT d'Aix-en-Provence - Département Informatique
Avenue Gaston Berger, F-13625 Aix-en-Provence, Cedex 1, France
laporte@romarin.univ-aix.fr

[2] LaBRI, CNRS UMR 5800 - Université de Bordeaux 1, Bât A30
351 Cours de la Libération, F-33405 Talence Cedex, France
novelli@labri.fr

[3] LIF Marseille, CNRS FRE-2504 - Université de la Méditerranée, Case 901
163 Avenue de Luminy, F-13288 Marseille Cedex 9, France
{cicchetti,lakhal}@lif.univ-mrs.fr

Abstract. In this paper, we propose a sound approach and an algorithm[1] for computing a condensed representation of either full or iceberg datacubes. A novel characterization of datacubes based on dimensional-measurable partitions is introduced. From such partitions, iceberg cuboids are achieved by using constrained product linearly in the number of tuples. Moreover, our datacube characterization provides a loss-less condensed representation specially suitable when considering the storage explosion problem and the I/O cost. We show that our algorithm Ccube turns out to an operational solution more efficient than competive proposals. It enforces a lecticwise and recursive traverse of the dimension set lattice and takes into account the critical problem of memory limitation. Our experimental results shows that Ccube is a promising candidate for scalable computation.

1 Motivation

Answering efficiently OLAP queries requires to pre-compute their results, i.e. datacubes [8], and to store them. Computing datacubes is specially costly in execution time [1,15,3] and preserving them is memory consuming because of the disk explosion problem [11].
Althrough intrinsically related, the two issues have been addressed separately. On one hand, various algorithms have been defined to compute datacubes more and more efficiently [1,15]. On the other hand, approaches have been proposed

[1] This work is partially supported by the AS CNRS-STIC "Data Mining"

to minimize storage requirements. They are based on physical techniques [17], choice of results to be materialized [11], or approximation model [2].
Recently the algorithm BUC was introduced [3]. It addresses the twofold issue of computing and storing cubes by taking into consideration the relevance of results. BUC aims to compute iceberg datacubes which are similar to "multi-feature cubes" proposed in [16]. When computing such cubes, aggregates not satisfying a selection condition specified by user (similar to the clause Having in SQL) are discarded. Let us notice that the algorithm H-Cubing is intended for computing iceberg cubes by enforcing more complex measures [10]. Motivations behind computing iceberg cubes are the following. Firstly, iceberg cubes provide users only with relevant results because scarce or exceptional dimensional combinations of values are discarded. Computation performances can be improved since the lattice to be explored can be pruned (using the selection condition) [3], and storage requirement is decreased. Precomputed iceberg cubes also offer an efficient solution for answering iceberg queries [5]. Another important issue, when computing iceberg cubes, concerns OLAP mining, i.e. the discovery of multidimensional association rules [9], classification rules [13], or multidimensional constrained gradients [4].

In this paper, we propose an approach for achieving full or iceberg datacubes. The originality of our proposal is that it aims to compute a loss-less condensed representation of datacubes which is specially less voluminous than the classical representation of the datacube.
The main contributions of our approach are the following. Firstly, we introduce a novel and sound characterization of datacubes based on the concept of dimensional-measurable partitions, inspired from the partition model [18]. From a dimensional-measurable partition, according to a set of dimensional attributes, computation of the associated cuboid is simple. A cuboid results from a single group-by query according to a certain set of dimensions [9]. This new concept is attractive because dealing with dimensional-measurable partitions means operating linearly set intersections (and thus the use of sorting or hash-based methods is avoided). Secondly, our characterization provides a condensed representation of datacubes in order to minimize disk storage and I/O cost. The third contribution provides a new principle, dictated by the critical problem of main memory limitation, for navigating through the dimensional lattice. It is called lecticwise and recursive traverse of the dimensional lattice, offers a sound basis for our computing strategy, and applies for computing full or iceberg datacubes. Finally, the described ideas are enforced through a new operational algorithm, called Ccube. Ccube has been experimented by using various benchmarks. Our experiments show that Ccube is more efficient than BUC.
The rest of the paper is organized as follows. In section 2, we introduce the novel concepts of our approach and characterize datacubes. Section 3 is devoted to our algorithmic solution and Section 4 to experiments. Finally, we discuss the strengths of our approach and evoke further research work.

2 Condensed Representation of Iceberg Datacubes

In this section, we introduce a novel characterization of datacubes which is based on simple concepts. It offers a condensed representation which can be seen as a logical and loss-less proposal for minimizing the storage explosion.
First of all, we assume that the relation r to be aggregated is defined over a schema R encompassing, apart from the tuple identifier, two kinds of attributes: (i) a set Dim of dimensions which are the criteria for further analysis, and (ii) a measurable attribute M standing for the measure being analyzed[2]. We also make use of the following notations: X stands for a set of dimensions $\{A_1, A_2 \ldots\}$, $X \subseteq Dim$. We assume that an anti-monotonic constraint w.r.t. inclusion $Cond$ is given by the user as well as an additive aggregative function f (e.g. *count*, *sum* ...).
Inspired from the concept of partition defined in [18], we introduce the concept of Dimensional-Measurable partition according to a set of dimensions.

Definition 1 Dimensional-Measurable Classes
Let r be a relation over the schema $R = (RowId, Dim, M)$, and $X \subseteq Dim$. The Dimensional-Measurable equivalence class of a tuple $t \in r$ according to X is denoted by $[t]_X$, and defined as follows:
$[t]_X = \{\ (u[RowId], u[M]) \ / \ u \in r, u[X] = t[X]\ \}$.

Example 1 Let us consider the classical example of studying the sales of a company according to various criteria such as the sold product (Product), the store (Store) and the year (Year). The measure being studied according to the previous criteria is the total amount of sales (Total). An instance of our relation example is illustrated in figure 1.
The DM-equivalence class of the tuple t_1, i.e. having RowId = 1, according to the dimension Product, groups all the tuples (their identifier and measurable value) concerning the product 100:
$[t_1]_{Product} = \{$ (1, 70) (2, 85) (3, 105) (4, 120) (5, 55) (6, 60) $\}$. $\square$

A Dimensional-Measurable partition (or DM-partition) of a relation, according to a set of dimensions, is the collection of DM-equivalence classes obtained for the tuples of r and satisfying the anti-monotonic constraint.

Definition 2 DM-Partition
Let r be a relation over the schema R, X a set of dimensions, $X \subseteq Dim$ and $Cond$ an anti-monotonic constraint. The DM-partition of r according to X is denoted by $\Pi_X(r)$, and defined as follows: $\Pi_X(r) = \{\ [t]_X \models Cond \ / \ t \in r\ \}$.

Example 2 Let us resume our relation example given in figure 1. We assume that the condition is: $Sum(Total) > 220$. The DM-partition, according to the attribute Product is given below (the various DM-equivalence classes are

[2] All definitions, given in this section, can be easily extended in order to consider a set of measures, like in [16].

Sales				
RowId	Product	Store	Year	Total
1	100	a	1999	70
2	100	a	2000	85
3	100	b	1999	105
4	100	b	2000	120
5	100	c	1999	55
6	100	c	2000	60
7	103	a	1999	36
8	103	a	2000	37
9	103	b	1999	55
10	103	b	2000	60
11	103	c	1999	28
12	103	c	2000	30

Fig. 1. The relation example Sales

delimited by <>). $\Pi_{Product}(Sales) = \{<(1,70)(2,85)(3,105)(4,120)(5,55)(6,60)>, <(7,36)(8,37)(9,55)(10,60)(11,28)(12,30)>\}$. □

Let us underline that our implementation of DM-partitions only preserves tuple identifiers which are used for indexing measurable values. In order to efficiently handle DM-partitions, we introduce the concept of constrained product.

Lemma 1 Constrained Product of DM-Partitions
Let r be a relation, and $\Pi_X(r)$, $\Pi_Y(r)$ two DM-partitions computed from r according to X and Y respectively. The product of the two partitions, denoted by $\Pi_X(r) \bullet_c \Pi_Y(r)$, is obtained as follows, and equal to $\Pi_{X \cup Y}$: $\Pi_X(r) \bullet_c \Pi_Y(r) = \{\ [t]_X \cap [t]_Y \models Cond\ /\ [t]_X \in \Pi_X(r),\ [t]_Y \in \Pi_Y(r)\ \} = \Pi_{X \cup Y}(r)$

Example 3 Let us consider $\Pi_{Product}(Sales)$ and $\Pi_{Store}(Sales)$. Their product, obtained by intersection of the associated DM-equivalence classes, is achieved and classes not respecting the condition are discarded. The result is the following: $\Pi_{Product}(Sales) \bullet_c \Pi_{Store}(Sales) = \{<(3,105)(4,120)>\}$. □

From any DM-partition, an iceberg cuboid can be simply achieved by aggregating the measurable values of its equivalence classes. Moreover the whole relevant information contained in a relation can be represented through the set of DM-partitions computed according to each dimensional attribute of its schema. These partitions are called, in the rest of the paper, the *original DM-partitions*. Provided with such a set of DM-partitions, it is possible to compute iceberg cuboids according to any dimension combination, by making use of constrained product of DM-partitions. Thus the iceberg datacube, derived from r, can be achieved from the original DM-partitions.

In our approach, an iceberg cuboid results from applying an aggregative function f over the measurable attribute M for each class in the DM-partition $\Pi_X(r)$. Each class is symbolized by one of its tuple identifiers.

Definition 3 Condensed Iceberg Cuboids
Let $\Pi_X(r)$ be a DM-partition of r, and f an aggregative function. For each equivalence class in $\Pi_X(r)$, $[t]_X.M$ stands for the set of values of the measurable attribute and $t[RowId]$ a tuple identifier from $[t]_X$. The cuboid aggregating values of M in r according to X is denoted by $Cuboid_X(r)$ and defined as follows:
$Cuboid_X(r) = \{\ (t[RowId],\ f([t]_X.M)\ )\ /\ [t]_X \in \Pi_X(r)\ \}$

Example 4 Let us consider the cuboid yielded by the following SQL query:
SELECT Product, Sum(Total) FROM Sales
GROUP BY Product HAVING Sum(Total) ¿ 220
Since the DM-partition $\Pi_{Product}(Sales)$ encompasses two DM-equivalence classes satisfying the anti-monotonic constraint, our condensed representation of the associated iceberg cuboid groups only two couples (delimited by <>): $Cuboid_{Product}(Sales) = \{< 1,495 >, < 7,246 >\}$. The former couple gives the sale amount for the product 100 (for all stores and years) and the latter provides a similar result for the product 103. □

For characterizing cuboids, we state an equivalence between our representation and the result of the aggregate formation defined by A. Klug [12].

Lemma 2 Correctness
Let $Cuboid_X(r)$ be a cuboid according to X, achieved from r by applying the previous definition with $Cond = true$. Then we have:

$$\begin{aligned} Aggregation < X, f > (r) &= \{\ t[X] \circ y / t \in r, y = f(\{t'/t' \in r, t'[X] = t[X]\})\ \}^3 \\ &= \{\ t[X] \circ f([t]_X.M) / \exists\ [u]_X \in \Pi_X(r) \\ &\qquad \textit{such that } (t[RowId], t[M]) \in [u]_X\ \} \end{aligned}$$

Definition 4 Condensed Iceberg Datacubes
Let r be a relation. The condensed iceberg datacube associated to r, noted $CUBE(r)$, is defined by:
$CUBE(r) = \{Cuboid_X(r) \neq \emptyset / X \in 2^{Dim}\}$ *where 2^{Dim} is the power set of Dim.*

Example 5 Let us consider the following query yielding the iceberg cube according to Product and Store (depicted in figure 2 (left)):
SELECT Product, Store, Sum(Total) FROM Sales
CUBE BY Product, Store HAVING Sum(Total) ¿ 220
The single equivalence class in $\Pi_\emptyset(Sales)$ satisfies the given condition (Sum(Total) = 741). When achieving the constrained product of the two DM-partitions given below, the condition is applied and the result is as follows: $\Pi_{Product}(Sales) \bullet_c \Pi_{Store}(Sales) = \{< (3,105)(4,120) >\}$. It is used for building a

^{3} Definition given by A. Klug in [12].

Sales by Product and Store		
Product	Store	Total
ALL	ALL	741
100	ALL	495
103	ALL	246
ALL	a	228
ALL	b	340
100	b	225

Sales by Product and Store	
$Cuboid_{\emptyset}$ =	$\{ < 1, 741 > \}$
$Cuboid_{Product}$ =	$\{ < 1, 495 >, < 7, 246 > \}$
$Cuboid_{Store}$ =	$\{< 1, 228 >, < 3, 340 > \}$
$Cuboid_{Product,Store}$ =	$\{< 3, 225 > \}$

Fig. 2. An iceberg cube example

tuple of the iceberg cuboid ($< 3, 225 >$) which carries the following information: the total amount of sales for the product and the store referred to in tuple t_3 is 225. Figure 2 (right) illustrates the condensed representation of the iceberg datacube example. □

3 Computing Condensed Iceberg Cubes

In this section, we give the foundations of our algorithmic solution. We begin by recalling the definition of the lectic order (or colexicographical order) [7]. Then, we propose a new recursive algorithm schema for enumerating constrained subsets of 2^{Dim} according to the lectic order.

Definition 5 Lectic Order
Let $(R, <)$ be a totally ordered and finite set. We assume for simplicity that R can be defined as follows: $R = \{1, 2, \ldots, n\}$. R is provided with the following operator: $Max : 2^R \rightarrow R$
$X \rightarrow$ *the last element of* X.
The lectic order $<_l$ is defined as follows:
$\forall X, Y \in 2^R$, $X <_l Y \Leftrightarrow Max(X - Y) < Max(Y - X)$.

This definition yields a strict linear order on the set of all subsets, called lectic order.
Example 6 Let us consider the following totally ordered set: $R = \{A, B, C, D\}$. Enumerating combinations of 2^R with respect to the lectic order provides the following result: $\emptyset <_l A <_l B <_l AB <_l C <_l AC <_l BC <_l ABC <_l D <_l AD <_l BD <_l ABD <_l CD <_l ACD <_l BCD <_l ABCD$. □

Proposition 1. *[6]*
$\forall X, Y \in 2^R$, $X \subset Y \Rightarrow X <_l Y$.

The previous proposition ensures that minimal subsets not respecting the anti-monotonic constraint (called negative border [14]) are the first ones encountered in the lectic order.

Let us underline that traverses following from lectic or lexicographical orders are both depth-first-search. However the lectic order is compatible with the anti-monotonic constraint whereas lexicographical order is not.

Recursive Algorithm Schema for Constrained Subset Enumeration in Lectic Order

The novel algorithm *LCS* (*Lectic Constrained Subset*) gives the general algorithmic schema used by CCUBE. It is provided with two dimensional attribute subsets X and Y, and handles a set of attribute combinations: the negative border (*NegBorder*). The latter set encompasses all the minimal attribute subsets not satisfying the anti-monotonic constraint. The algorithm is based on a twofold recursion and the recursive calls form a binary tree in which each execution branch achieves a dimensional subset. The general strategy for enumerating dimensional attribute subsets consists of generating firstly all the constrained subsets (i.e. satisfying *Cond*) not encompassing a dimensional attribute, and then all the subsets which encompass it. More precisely, the maximal attribute, according to the lectic order, is discarded from Y and appended to X in the variable Z. The algorithm is recursively applied with X and the new subset Y. If Z is not a superset of an element in $NegBorder$, then the algorithm assesses whether the condition holds for Z. If it does, the algorithm is recursively called with the parameters Z and Y, else Z is added to $NegBorder$. The first call of *LCS* is provided with the two parameters $X = \emptyset$ and $Y = Dim$ while $NegBorder$ is initialized to $\{\}$.

```
Algorithm $LCS( X, Y )$
1  if $Y = \emptyset$ then output $X$
2  else
3      $A := Max(Y)$
4      $Y := Y - \{A\}$
5      $LCS(X, Y)$
6      $Z := X \cup \{A\}$
7      if $\nexists\ T \in NegBorder$ such that $T \subseteq Z$
8      then
9          if $Z \models Cond$
10         then $LCS(Z, Y)$
11         else $NegBorder := NegBorder \cup Z$
```

Lemma 3 Algorithm Correctness
The correctness of the algorithm LCS is based on proposition 1 and:

(i) due to the anti-monotonic property of Cond, we have:
$\forall\ X \subseteq Y,\ Y \models Cond \Rightarrow X \models Cond$;

(ii) from the distributivity property of the dimensional lattice, we have: $\forall\ A \in Dim,\ \forall\ X \subset Dim,\ 2^{X \cup \{A\}} \cap 2^{X - \{A\}} = \emptyset$. *Thus any dimensional subset is enumerated only once.*

Example 7 Let us consider our relation Sales and the condition $Sum(Total) \geq 350$. In this context, the binary tree of recursive calls when running our algorithm is depicted in figure 3. Leaves in this tree correspond to outputs which are, from left to right, ordered in a lectic way. In any left subtree, all subsets not encompassing the maximal attribute (according to the lectic order) of the subtree root are considered while in right subtrees, the maximal attribute is preserved. □

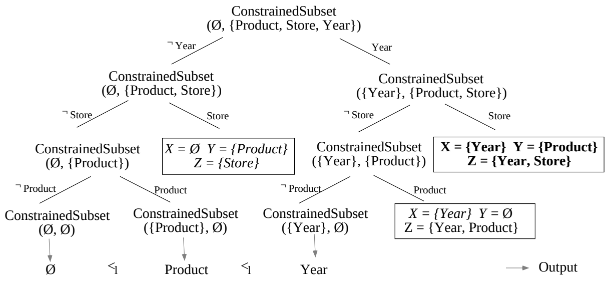
ConstrainedSubset (Ø, {Product, Store, Year})
¬ Year
Year
ConstrainedSubset (Ø, {Product, Store})
ConstrainedSubset ({Year}, {Product, Store})
¬ Store
Store
¬ Store
Store
ConstrainedSubset (Ø, {Product})
X = Ø Y = {Product} Z = {Store}
ConstrainedSubset ({Year}, {Product})
X = {Year} Y = {Product} Z = {Year, Store}
¬ Product
Product
¬ Product
Product
ConstrainedSubset (Ø, Ø)
ConstrainedSubset ({Product}, Ø)
ConstrainedSubset ({Year}, Ø)
X = {Year} Y = Ø Z = {Year, Product}
Ø
<l
Product
<l
Year
Output

Fig. 3. Execution tree

We propose an algorithm, called Ccube, for computing full and iceberg datacubes. It fits in the theoretical framework previously presented. A pre-processing step is required in order to build DM-partitions according to each single attribute from the input relation. According to the user need, these DM-partitions can be full or iceberg partitions. While performing this initial step, the computation of the cuboid according to the empty set is operated and its result is yielded.
If the original partitions ($\cup_{A \in Dim} \Pi_A(r)$) cannot fit in main memory, then the fragmentation strategy proposed in [15] and used in [3] is applied. It divides the input relation in fragments according to an attribute until the original associated DM-partitions can be loaded. Ccube adopts the general algorithm schema described through LCS but it is intended to compute all desired aggregates and thus it yields the condensed representation of all possible cuboids. Ccube deals with DM-partitions and enforces constrained product of DM-partitions. Like LCS, its input parameters are the subsets X and Y. When Z is not a superset of a negative border element, its DM-partition is computed by applying the constrained product of the in-memory DM-partitions $\Pi_X(r)$ and $\Pi_A(r)$. The constrained product is implemented through two functions called *Product* and *Prune*. The latter discards DM-equivalence classes not satisfying the anti-monotonic constraint and the second recursive call is performed only if the DM-partition according to Z is not empty. The $NegBorder$ encompasses the minimal attribute combinaisons Z such that $Cuboid_Z(r) = \emptyset$. The pseudo-code of the algorithm Ccube is given below.

```
**Algorithm** $CCUBE( X, Y )$
1  **if** $Y = \emptyset$ **then** Write $Cuboid_X(r)$
2  **else**
3      $A := Max(Y)$
4      $Y := Y - \{A\}$
5      $CCUBE(X, Y)$
6      $Z := X \cup \{A\}$
7      **if** $\nexists\, T \in NegBorder$ such that $T \subseteq Z$
8      **then**
9          $\Pi_Z(r) := Product( \Pi_X(r), \Pi_A(r) )$
10         $\Pi_Z(r) := Prune( \Pi_Z(r) )$
11         **if** $\Pi_Z(r) <> \emptyset$
12         **then** $CCUBE(Z, Y)$
13         **else** $NegBorder := NegBorder \cup Z$
```

4 Experimental Comparison

In order to assess performances of Ccube, the algorithm was implemented using the language C++. An executable file can be generated with Visual C++ 5.0 or GNU g++ compilers. Experiments were performed on a Pentium Pro III/700 MHz with 2 GB, running Linux.

The benchmark relations used for experiments are synthetic data sets automatically generated under the assumption that the data is uniformly and at random distributed. With these benchmarks, optimization techniques (such as attribute ordering used in BUC) do not improve efficiency.

An executable version of BUC is not available from the authors, we have therefore developed a new version of this algorithm under the very same conditions than for Ccube implementation and with a similar programming style.

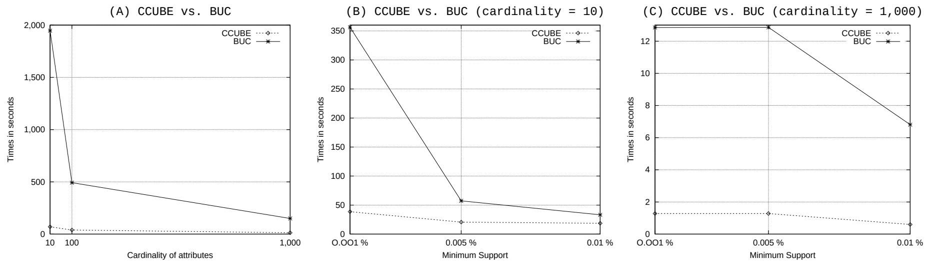

Fig. 4. Execution times in seconds for relations with 100,000 tuples and 10 attributes

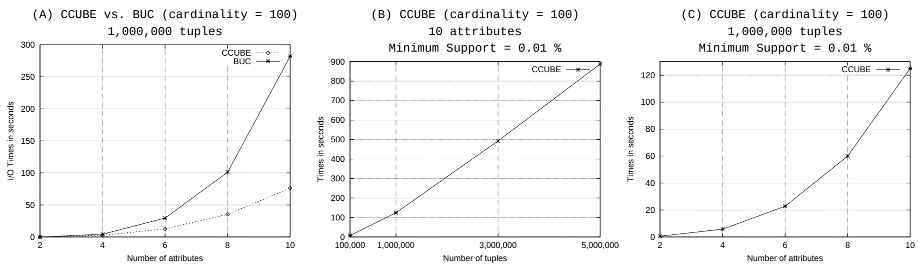

Fig. 5. I/O times in seconds for various numbers of attributes (A) and execution times for various numbers of tuples and attributes with a minimum support (B, C)

Figure 4 (A) gives the execution times of the two algorithms, when computing a fullcube and varying the dimension cardinalities from 10 to 1,000. The input is a relation encompassing 100,000 tuples and 10 attributes. As expected, CCUBE behaves specially well. As mentioned in [3], BUC is penalized when domain cardinalities are small and the gap between execution times of CCUBE and BUC decreases as the domain cardinality increases. Figures 4 (B) and (C) provide execution times of CCUBE and BUC when computing the iceberg cube from the same input relation. The minimum support varies from 0.001 % to 0.01 % and the used function is *Count*. The cardinality of all dimensions is set to 10 or 1,000. In any case, CCUBE is more efficient than BUC.

Figure 5 (A) gives the times required for writing results. The input relation encompasses 1,000,000 tuples and the number of dimensions varies from 2 to 10. Dimension cardinality is set to 100.

The curves, in figure 5 (B, C), illustrate CCUBE scalability according to the tuple number and dimension number when computing an iceberg datacube with a minimum support set to 0.01 %. The tuple number of the input relation varies from 100,000 to 5,000,000. As expected, CCUBE behaves linearly in the number of tuples. The dimension number varies from 2 to 10.

5 Conclusion

The approach presented in this paper addresses the computation of either full or iceberg datacubes. It fits in a formal framework proved to be sound and based on simple concepts. We propose an alternative representation of data sets to be aggregated: the DM-partitions. By selecting relevant DM-partitions, we show that on one hand memory requirement is decreased when compared to BUC one [3], and on the other hand all necessary cuboids can be computed by enforcing in-memory DM-partition products, i.e. by performing set intersections, linearly in the set cardinalities. CCUBE traverses the dimensional lattice by following from the lectic order. Its navigation principles are soundly founded. In addition, we propose a condensed representation of datacubes which significantly reduces the necessary storage space without making use of physical techniques. We show that CCUBE has good scale-up properties and is more efficient than BUC. Intended further work concerns an extension of datacubes that we call decision datacubes which represent small covers for associative classification rules.

References

1. S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the Computation of Multidimensional Aggregates. In *VLDB'96*, pages 506–521, 1996.
2. D. Barbará and M. Sullivan. Quasi-Cubes: Exploiting Approximations in Multidimensional Databases. *SIGMOD Record*, 26(3):12–17, 1997.
3. K.S. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs. In *ACM SIGMOD, USA*, pages 359–370, 1999.
4. G. Dong, J. Han, J. M. W. Lam, J. Pei, and K. Wang. Multi-Dimensional Constrained Gradients in Data Cubes. In *VLDB'01*, pages 321–330, Italy, 2001.
5. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman. Computing Iceberg Queries Efficiently. In *VLDB'98, New York City, New York, USA*, pages 299–310. Morgan Kaufmann, 1998.
6. B. Ganter and K. Reuter. Finding all Closed Sets: A General Approach. *Order*, 8:283–290, 1991.
7. B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations.* Springer-Verlag, 1999.
8. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals. *Data Mining and Knowledge Discovery*, 1(1), 1997.
9. J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2001.
10. J. Han, J. Pei, G. Dong, and K. Wang. Efficient Computation of Iceberg Cubes with Complex Measures. In *ACM SIGMOD'01*, USA, 2001.
11. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD'96*, pages 205–216, Montreal, Quebec, Canada, June 1996.
12. A. C. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of ACM*, 29(3):699–717, 1982.
13. H. Lu and H. Liu. Decision Tables: Scalable Classification Exploring RDBMS Capabilities. In *VLDB'00*, pages 373–384, Cairo, Egypt, September 2000.
14. H. Mannila and H. Toivonen. Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Mining and Knowledge Discovery*, 10(3):241–258, 1997.
15. K.A. Ross and D. Srivastava. Fast Computation of Sparse Datacubes. In *VLDB'97, Athens, Greece*, pages 116–125, 1997.
16. K.A. Ross, D. Srivastava, and D. Chatziantoniou. Complex Aggregation at Mutiple Granularities. In *EDBT'98*, LNCS vol. 1377, pages 263–277. Springer Verlag, 1998.
17. K.A. Ross and K.A. Zaman. Serving Datacube Tuples from Main Memory. In *SSDM'2000, Berlin, Germany*, pages 182–195, 2000.
18. N. Spyratos. The partition model: A deductive database model. *ACM TODS*, 12(1):1–37, 1987.