On the Computation of Maximal-Correlated Cuboids Cells

Ronnie Alves[*] and Orlando Belo

Department of Informatics, School of Engineering, University of Minho
Campus de Gualtar, 4710-057 Braga, Portugal
{ronnie, obelo}@di.uminho.pt

Abstract. The main idea of iceberg data cubing methods relies on optimization techniques for computing only the cuboids cells above certain minimum support threshold. Even using such approach the curse of dimensionality remains, given the large number of cuboids to compute, which produces, as we know, huge outputs. However, more recently, some efforts have been done on computing only *closed cuboids*. Nevertheless, for some of the dense databases, which are considered in this paper, even the set of all closed cuboids will be too large. An alternative would be to compute only the *maximal cuboids*. However, a pure maximal approaching implies loosing some information, this is one can generate the complete set of cuboids cells from its maximal but without their respective aggregation value. To play with some "loss of information" we need to add an interesting measure, that we call the *correlated value of a cuboid cell*. In this paper, we propose a new notion for reducing cuboids aggregation by means of computing only the *maximal-correlated cuboids cells*, and present the M3C-Cubing algorithm that brings out those cuboids. Our evaluation study shows that the method followed is a promising candidate for scalable data cubing, reducing the number of cuboids by at least an order of magnitude or more in comparison with that of closed ones.

1 Introduction

Efficient computation of data cubes has been one of the focusing points in research since the introduction of data warehousing, OLAP, and the data cube operator [8]. Data cube computation can be formulated as a process that takes a set of tuples as input, computes with or without some auxiliary data structure, and materializes the aggregated results for all cells in all cuboids. Its size is usually much larger than the input database, since a table with n dimensions results in 2^n cuboids. Thus, most work is dedicated to reduce either the computation time or the final cube size, such as efficient cube computation [3, 14, 9], or cube compression [11, 15]. These cost reduction processes are all without loss of any information, while some others, like the approximation [1] or the iceberg-cube [5, 3, 14, 9] ones, reduce the costs by skipping trivial information.

[*] Supported by a Ph.D. Scholarship from FCT-Foundation of Science and Technology, Ministry of Science of Portugal.

The ideas of compressing cuboids cells in terms of classes of cells, or closed cells, seem to be an interesting approach to reduce size complexity, and also to explore optimally the semantics from the cube lattice. Nevertheless, for some of the dense databases we consider in this paper, even the set of all closed cuboids cells would grow to be too large. The only recourse may be to mine the maximal cuboids cells in such domains. However, a pure maximal approaching implies loosing some information - one can generate the complete set of cuboids cells from its maximal but without their respective aggregation value. To play with some loss of information we propose a new measure, that we called the *correlated value of a cuboid cell.* This measure is inspired on *all_confidence* measure [12], which has been successfully adopted for judging interesting patterns in association rule mining, and further exploited with confidence closed correlated pattern [10]. This measure must disclose true correlation (also dependence) relationship among cuboids cells and needs to hold the null-invariance property. Furthermore, real world databases tend to be correlated, i.e., dimensions values are usually dependent on each other. The main motivation of the proposed method emerged from the observation that real databases tend to be correlated, i.e., dimensions values are usually dependent on each other. For example, Store “Wallgreens” always sells Product “Nappy” or Store “Starbucks” always makes Product “Coffee”. In addition, the result of correlated cells on the corresponding data cube is to generate a large number of cells with same aggregation values. The Range CUBE method was the first approach to explore correlation among dimensions values by using a range trie [6]. Although it does not compress the cube optimally and may not disclose true correlation relationship among cuboids cells holding the null-invariance property [12, 16, 10]. Inspired on the previous issues we raise a few questions to drive this work:

1. *Can we develop an algorithm which captures maximal correlated cuboids cells on dense/sparse databases?*
2. *How much such an approach can reduce the complete set of cuboids in comparison with the other approaches (i.e., pure maximal to closed ones)?*
3. *How about the data cubing costs?*

In this paper, we propose a new iceberg cube mining method for reducing cuboids aggregation by means of computing only the *maximal-correlated cuboids cells*, and present the M3C-Cubing algorithm that brings out those cuboids.

2 Maximal-Correlated Cuboids Cells

A cuboid is a multi-dimensional summarization of a subset of dimensions and contains a set of cells. A data cube can be viewed as a lattice of cuboids, which also integrates a set of cells.

Definition 1 – Cuboid Cell – In an n-dimension data cube, a cell $c = (i_1,i_2,\ldots,i_n : m)$ (where m is a measure) is called a k-dimensional cuboid cell (i.e., a cell in a k-dimensional cuboid), if and only if there are exactly k $(k \leq n)$ values among $\{i_1,i_2,\ldots,i_n\}$ which are not * (i.e., all). We further denote $M(c) = m$ and $V(c) = (i_j,i_2,\ldots,i_n)$. In this paper, we assume that the measure *m* is count.

A cell is called *iceberg cell* if it satisfies a threshold constraint on the measure. For example, an iceberg constraint on measure count is *M(c)* $\geq$ *min_supp* (where *min_supp* is a user-given threshold). Given two cells c = ($i_1,i_2,\ldots,i_n$: m) and c' = ($i'_1,i'_2,\ldots,i'_n$: m'), we denote V(c) $\leq$ V(c') if for each i_j (j = 1,…,n) which is not *, $i'_j = i_j$. A cell c is said to be covered by another cell c' if for each c'' such that V(c) $\leq$ V(c'') $\leq$ V(c'), M(c'') = M(c'). A cell is called a *closed cuboid cell* if it is not covered by any other cells. A cell is called a *maximal cuboid cell* if it is closed and has no other cell c which is superset of it (we have an *exception* just in case when its correlated value is higher than a minimum threshold).

Definition 2 – The Correlated Value of a Cuboid Cell – Given a cell c, the correlated value *3CV* of a V(c) is defined as,

$\text{maxM}(c) = \max \{M(c_i) | \text{for each } c_i \in V(c)\}$ Eq.(1)

$3CV(c) = M(c) / \text{maxM}(c)$ Eq.(2)

Definition 3 – Maximal Correlated Cuboid Cell – A cell c is a *maximal-correlated cuboid cell* (M3C) if it is covered by a *maximal cuboid cell*, its M(c) value is higher than *min_supp* and its *3CV(c)* value is higher than *min_3CV* (where *min_3CV* is a user-given threshold for correlation)

From the last definition we allow a *correlated exception* for its supersets, where it is true when cell c is covered by another cell c' and *3CV(c')* is higher than *min_3CV.*

Given the above definitions, the problem of computing the *maximal-correlated cuboids cells* is to compute all maximal cuboids cells which satisfy iceberg constraints and its *correlated exception* cells. An example of the maximal-correlated cuboids cells is given in Example 1.

Table 1. Example of Maximal Correlated Cuboids Cells

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a1	b1	c1	d3
a1	b1	c1	d1
a2	b2	c2	d4

Example 1 - Maximal Correlated Cuboids Cells. Table 1 shows a table (with four attributes) in a relational database. Let the measure be count, the iceberg be count $\geq$ 2 and the correlated value 3CV $\geq$ 0.85. Then c1 = (a1,b1,c1,* : 3) and c2 = (a1,*,*,* : 4) are closed cells; c1 is a maximal cell; c3 = (a1,b1,*,* : 3) and c4 = (*,b1,c1,* : 3) are covered by c1; but c4 has a correlated exception (3CV=1); c5 = (a2,b2,c2,d4 : 1) does not satisfy the iceberg constraint. Therefore, c1 and c4 are maximal correlated cuboids' cells. The iceberg condition is *count* $\geq$ *min_sup* and *the correlated exception value* $\geq$ *min_3CV.*

3 M3C-Cubing

The proposed method for extraction of the Maximal-Correlated Cuboids Cells follows the BUC data cubing ideas [3] – we call it as M3C-Cubing. The computation starts from the smallest cuboids of the lattice, and works its way towards the larger, less aggregated cuboids. Our method does not share the computation of aggregates between parent and child cuboids, only the partitioning cost. Besides, as was verified by BUC experimental results, partitioning is the major expense, not the aggregation one.

We begin by defining the necessary terminology for describing the M3C-Cubing algorithm. We consider a *base relation cell* to be a mapping *K(c)→ M(c)*, where *K* is a composite key built from the grouping attributes values in V(c) and *M(c)* is also a composite key with the value to be aggregated. From the base relation cell we can extract several partitions; each partition has a subset of cells to aggregate. The *partition* of a base relation cell is defined as *P(c)→ {K(c)→M(c)}*, where *P(c)* is the partition key.

In order to get the *correlated value of a cuboid cell (3CV)* we need to keep the aggregation value for each *1-D* cuboid. This is denoted as a mapping from *1-D(c)→ M(c)*. We should note that the maximum value will occur when the subset *K(c)* consists of a single grouping attribute (see *Definition 2*).

M3C-Cubing is guided by an SE-tree framework, first introduced by Rymon [13], and adopted later by Mafia [4] and Max-miner[2]. In this work, we call as *M3C-tree*. The *M3C-tree* is traversed by using a *pure depth-first* (DFS) order. Each node of the *M3C-tree* provides *n-D* cuboids which will be further partitioned, aggregated and checked if it is maximal or not (see *Definition 1*). In general, superset pruning works better with DFS order since many least aggregated cuboids may already have been discovered.

The strategies for pruning non-maximal correlated cuboids cells (*nonM3C*) basically attempt to: test out iceberg condition, check if it is maximal and when is not, check if it is a *correlation exception* (see *Definition 3*). They are just discarded in case its *3CV* value is lower than a minimum threshold (*min_3CV*). Consequently, we provide the complete set of interesting cuboids which are maximal-correlated cuboids.

M3C-Cubing also keeps the *current cuboids* aggregated and the *previous one* for further pruning out of *nonM3C* cells. To speed up this process, we cannot remove an entire branch of the *M3C-tree*, since we have to aggregate its related partitions in order to validate the pruning conditions mentioned before. In this sense, we are just able to prune out *nonM3C* cells by the time we expand the *M3C-tree* level-by-level.

Algorithm 3.1. M3C-Cubing: Computing maximal-correlated cuboids cells

```
Input:    a table relation trel; min_supp; min_3CV.
Output  : the set of maximal-correlated cuboids cells.
Method  :
1. Let brel be the base relation of trel.
2. Build the M3C-tree concerning the grouping attributes in brel.
3. Call M3C-Cubing (min_supp, min_3CV, brel, M3C-tree).
```

```
Procedure M3C-Cubing (min_supp, min_3CV, brel, M3C-tree)
 1: get 1-D cuboids from M3C-tree
 2: for each j in 1-D cuboids do
 3:    get its partition from brel on dimensions [n], and
        set part ←{K(c)→M(c)}
 4:    aggregate part and set agg ← {V(c)→M(c)} when M(c)>=min_supp
 5:    set allCbs ← {agg}; set 3cv-1d ← {allCbs}
 6: end for
 7: get n-D cuboids in DFS order from M3C-tree
 8: for each k in n-D cuboids do
 9:    get its partition from brel on dimensions [n-D],
       and set part ←{K(c)→M(c)}
10:    aggregate part and set agg ← {V(c)→M(c)} when M(c)>=min_supp
11:    set allCbs ← allCbs  ∪ {agg}; set currCbs ← {agg}
12:    set 3cv-nd ← 3cv-nd  ∪ {call 3cv-nd(3cv-1d, agg )}
13:    set maxCbs ← maxCbs ∪
           {call maxCorr (allCbs, currCbs, 3cv-nd, min_3CV )}
14: end for
Procedure maxCorr(allCbs, currCbs, 3cv-nd, min_3CV)
  1: set nonM3C ←{ }
  2: for each j in currCbs do
  2:     for each k in allCbs do
  3:         if dom(allCbs) is superset of dom(currCbs),
              nonM3C ← nonM3C ∪ {dom(currCbs)}
  4:     end for
  5: end for
  6: remove any nonM3C in allCbs where 3cv-nd(nonM3C)< min_3CV
  7: return allCbs

Procedure 3cv-nd(3cv-1d, agg)
  1: for each j in agg do
  2:     splits into 1-D cells; maxValue=0;
  3:     for each k in 1-D cells do
  4:         get its aggValue(3cv-1d)
  5:         if aggValue>=maxValue, maxValue=aggValue
  6:     end for
  7:     set 3cv=SI{agg}/ maxValue; set 3cv-nd ← {dom(agg) → 3cv}
  8: end for; return 3cv-nd
```

With the aim of evaluating how M3C-Cubing reduces the final set of cuboids, we have to do a few modifications to the main method to support both pure maximal cuboids and closed ones. Those modifications are available as two new procedures: One for pure maximal and other for closed cuboids. We omit here those procedures, but one can also follows the definitions on section 2.

To bring out the pure maximal we just need to re-write the line 6 in *maxCorr* Procedure. Thus, the conditional test on *3CV* value of the ancestor cuboids is set apart. Needless to say, that we cannot make any use of *3CV-nd* procedure either to get pure maximal or closed cuboids. To get just the closed cuboids we must verify the *closedness* property [15] among the cuboids, consequently we just have to check if its not covered by other cells.

3.1 Cover Equivalence in M3C-Cubing

The idea of grouping cover partitions cells into classes can also be explored by M3C-Cubing in order to shrink even more the final data cube. By definition 1, it is possible to group a set of cuboids cells by verifying those cells which are cover equivalent ones [11]. Thus, these cells essentially have the same value for any aggregate on any measure but with different degrees of correlation. For instance, in Example 1 the cells (a1,b1,c1,*) and (*,b1,c1,*) are cover equivalent cells. Next, we present a few concepts for guiding the grouping cover partition process with M3C-Cubing.

Cover partitions – The partition induced by cover equivalence is convex. Cover partitions can be grouped into a M3C class (*ji..jn*) (Figure 1). Each class in a cover partition has a unique maximal upper bound, and a unique lower bound (Table 2).

Upper bound cell – The upper bound for a particular class is the maximal cuboid cell contained in this class. Such as, in Example 1, the cell (a1,b1,c1,*) is the upper bound cell.

Lower bound cell – The lower bound cell for a particular class is the maximal 3CV value achieved by the correlated value of the cuboid cell (see *Definition 2*). Since M3C cubing allows catching all correlated exception cuboids, the lower bound cell will be that one with the highest 3CV value. E.g., in Example 1, the lower bound cell for class *j1* is given by (*,b1,c1,*).

To explore maximal correlation over those classes we have to define the *local_3CV* value for each class. The *local_3CV* value of a class is the maximum local value given by the lower bound cell of each class (Table 2).

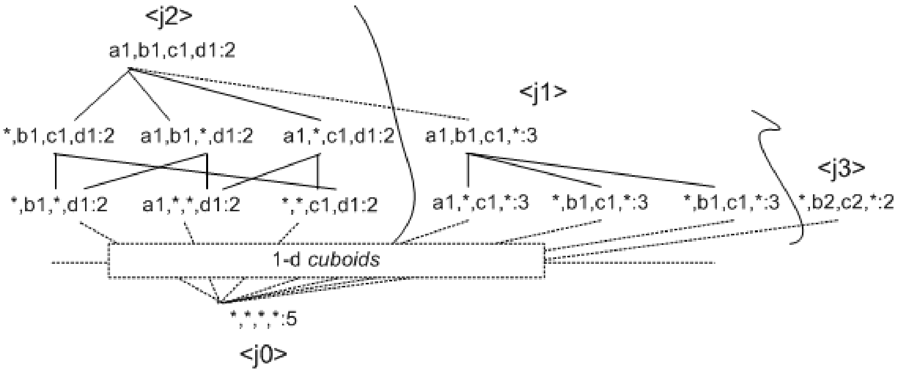

Fig. 1. The lattice formed by the cover partitions of the Example 1

Table 2. The set of classes from Example1

ClassID	UpBound	LoBound	Local_3CV	Lat_child	Agg
j0	(*,*,*,*)	(*,*,*,*)	0	-1	5
j1	(a1,b1,c1,*)	(*,b1,c1,*)	1	j0	3
j2	(a1,b1,c1,d1)	(*,b1,*,d1)	2/3	j0	2
j3	(*,b2,c2,*)	(*,b2,c2,*)	1	j0	2

4 Evaluation Study

In this section, we report our experimental results on the shrinking and performance aspects of each method (M3C=Maximal-Correlated, Max = pure Maximal and

Closed). The results are quite the same concerning to the performance point of view. This is true because those methods were developed having the basis on our main method (M3C). So, we can take those results only as an example of how much these modifications affect the whole time processing of the M3C-Cubing. On the other hand, reducing aspects of M3C-Cubing shows its viability by providing an interesting tradeoff between a pure maximal approaches to a closed one.

All the experiments were performed on a 3GHz Pentium IV with 1Gb of RAM memory, running Windows XP Professional. M3C-Cubing was coded with Java 1.5, and they were performed over eight synthetic datasets (Table 3). The values for columns *min_3CV%* and *min_supp* are provided for the tests when fixing one value and varying the other. The density column shows the degree(%)[1] of density/sparseness of each dataset. All datasets have a normal distribution.

Table 3. The overall information of each dataset

Dset	Tuples	Dims	Card.	Density	Min_3CV%	Min_Supp%
d1	100	3	3	27%	40%	2%
d2	250	5	3	97%	27%	0.80%
d3	500	3	5	25%	10%	0.60%
d4	750	4	5	83%	15%	0.67%
d5	1000	5	3	24%	17%	0.40%
d6	1250	4	6	100%	9%	0.16%
d7	3000	7	3	73%	35%	0.10%
d8	5000	6	4	82%	50%	0.04%

We first show that the complete set of maximal-correlated cuboids cells (M3C) is much smaller in comparison with both that of pure maximal (Max) and that of close ones (Closed). Figure 2 shows the number of cuboids generated by each approach from all datasets. The number of cuboids is plotted on a *log scale*. Figure 2(a) presents the number of cuboids generated when *min_3CV* is fixed and *min_supp*

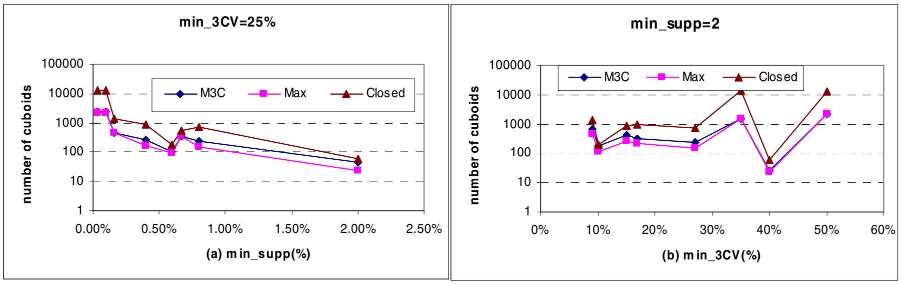

Fig. 2. Number of cuboids generated from all datasets

[1] The density degree of a dataset is calculated by the division: the product of each dimension (its cardinality value) by the number of tuples within the related dataset. The dataset is more dense when density degree is close to 100%.

varies, while figure 2(b) shows those cuboids the other way around, fixing *min_supp* and varying *min_3CV*. These figures show that the M3C generates much smaller cuboids under lower *min_supp* or lower *min_3CV*. They also illustrate how much bigger the closed cuboids are in comparison with the other two methods. These results also indicate that under higher density datasets the chances of reducing cuboids by M3C-Cubing is more effective. Furthermore, the distance between Max and M3C reveals the gap of cuboids which are discarded (higher correlated ones) when using a pure maximal approach.

The next two figures (figure 3(a) and figure 3(b)) show the performance aspects of M3C-Cubing from all datasets. These figures follow the same configuration properties from previous two (figure 2(a) and figure 2(b)). Figure 3(a) illustrate that under a fixed *min_supp*, the maximal-correlated cuboids are useful only with lower *3CV* thresholds. This is confirmed by the *downward property* [12] of *3CV_value* of a cuboid cell. By the time the data cubing process is getting closer to the least aggregated cuboids, the *3CV_value* also decreases, so the computation time is pretty close, because *3CV* is decreasing. Figure 3(b) points out the effectiveness of M3C-Cubing under lower *min_supp*, giving more likelihood to identify correlated-cuboids, increasing a little-bit the processing time to prune out *nonM3C* cells.

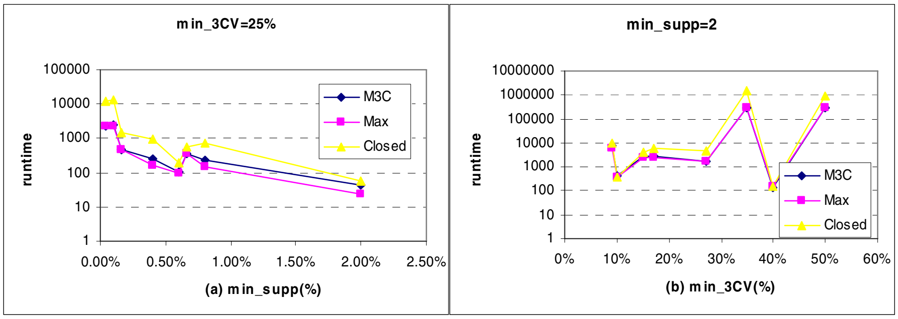

Fig. 3. The execution time from all datasets

Now, we are going to present a few examples concerning the reduction costs of computing the complete cube instead of a “partial” data cubing process. Figures ranging from figure 4(a) to figure 4(d) illustrate those results. Figure 4(a) and figure 4(b) shows the number of cuboids generated when *min_supp* varies and *min_3CV* is fixed, while figures 4(c), 4(d) present those when *min_3CV* varies and *min_supp* is fixed. Under any circumstances the final cube is quite closer to the closed one, which points out that even using such closed-reduction the cuboids size remains even bigger. It is also demonstrates how much M3C-Cubing can save in comparison with the other methods.

In summary, the experimental results show that the number of maximal-correlated cuboids is quite smaller in comparison with that of the closed ones. Even, with a few modifications to the main features of M3C-Cubing, it still performs competitively with the other ones. Moreover, we provide the set of all maximal-correlated cuboids.

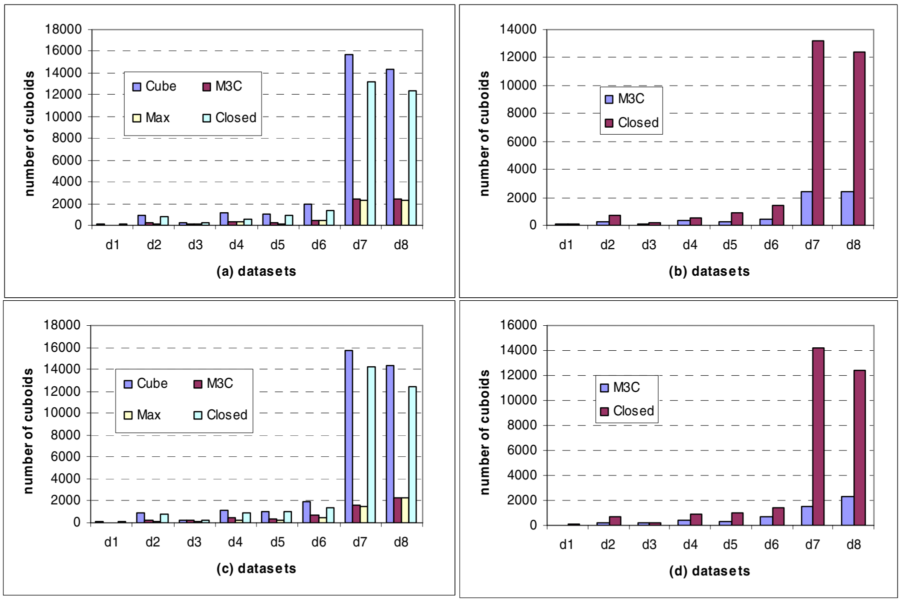

Fig. 4. Reductions cost of cubing over all datasets

5 Final Remarks

The motivation behind iceberg cube mining is tightly related to reducing the search space for computing aggregates. The classical methods either offer ways for sharing the computation among cuboids or for exploring the partition costs in order to reduce the large output size.

We have presented M3C-Cubing that effectively reduces the complete set of cuboids cells introducing a new notion called *maximal-correlated cuboids cells*. Through this cubing method we can find the maximal combinations among the grouping attributes and also keep its exceptional correlated cuboids, which can also indicates interesting changes on the cuboids during the cubing process.

We also have plans to investigate other aspects not addressed in this work such as: the application of *3CV* measure over other aggregate functions (average, min, max…) and the issues related to recover an aggregation value of subcells of an M3C cell.

For efficient mining of those cuboids we have devised M3C-Cubing which is guided by an *M3C-tree* with a pure DFS traversal order. In order to improve pruning, we must investigate the tail information of each node in the *M3C-tree* such as designed in [7, 17].

Finally, our evaluation study shows that maximal-correlated cuboids computation reduces the number of cuboids by at least an order of magnitude or more in comparison with the traditional approaches.

References

1. Barbara, D., Sullivan, M.: Quasi-cubes: Exploiting Approximations in Multidimensional Databases. In Proc. Int. Conference on Management of Data (SIGMOD), 1997.
2. Bayardo, R.: Efficiently Mining Long Patterns from Databases. In Proc. Int. Conference on Management of Data (SIGMOD), 1998.
3. Beyer, K., Ramakrishnan, R.: Bottom-up Computation of Sparse and Iceberg Cubes. In Proc. Int. Conference on Management of Data (SIGMOD), 1999.
4. Burdick, D., Calimlim, M., Gehrke, J.: MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In Proc. Int. Conference on Data Engineering (ICDE), pp.443-452, 2001.
5. Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R., Ullman, J., D.: Computing Iceberg Queries Efficiently. In Proc. Int. Conference on Very Large Databases (VLDB), 1998.
6. Feng, Y., Agrawal, D., Abbadi, A.-E., Metwally, A.: Range Cube: Efficient Cube Computation by Exploiting Data Correlation. In Proc. Int. Conference on Data Engineering (ICDE), 2004.
7. Gouda, K., Zaki, J.: GenMax : An Efficient Algorithm for Mining Maximal Frequent Itemsets. Data Mining and Knowledge Discovery, 11:1-20, 2005.
8. Gray, J., Bosworth, A., Layman, A., Pirahesh, A.: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In Proc. Int. Conference on Data Engineering (ICDE), 1996.
9. Han, J., Pei, J., Dong, G., Wank, K.: Efficient Computation of Iceberg Cubes with Complex Measures. In Proc. Int. Conference on Management of Data (SIGMOD), 2001.
10. Kim, W.-Y., Lee, Y.-K., Han, J.: CCMine: Efficient Mining of Confidence-Closed Correlated Patterns. In Proc. Int. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2004.
11. Lakshmanan, V.S., Pei, J., Han, J.: Quotient Cube: How to Summarize the Semantics of a Data Cube. In Proc. Int. Conference on Very Large Databases (VLDB), 2002.
12. Omiecinski. Alternative Interest Measures for Mining Associations. IEEE Trans. Knowledge and Data Engineering, 15:57-69, 2003.
13. Rymon, R.: Search through Systematic Set Enumeration. In Proc. Int. Conference on Principles of Knowledge Representation and Reasoning (KR), 539-550, 1992.
14. Shao, Z., Han, J., Xin, D.: MM-Cubing: Computing Iceberg Cubes by Factorizing the Lattice Space. In Proc. Int. Conference on Scientific and Statistical Database Management (SSDBM), 2004.
15. Xin, D., Han, J., Shao, Z., Liu, H.: C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking. In Proc. Int. Conference on Data Engineering (ICDE), 2006.
16. Xiong, H., Tan, P.-N., Kumar, V. Mining Strong Affinity Associations Patterns in Data Sets with Skewed Support Distribution. In Proc. Int. Conference on Data Mining (ICDM), 2003.
17. Zou, Q., Chu, W.-W., Lu, B.: SmartMiner: A Depth First Algorithm Guided by Tail Information for Mining Maximal Frequent Itemsets. In Proc. Int. Conference on Data Mining (ICDM), 2002.