C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking*

Dong Xin [1] Zheng Shao [1] Jiawei Han [1] Hongyan Liu [2] †
[1]University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
[2]Tsinghua University, Beijing, China, 100084

Abstract

It is well recognized that data cubing often produces huge outputs. Two popular efforts devoted to this problem are (1) iceberg cube*, where only significant cells are kept, and (2)* closed cube*, where a group of cells which preserve roll-up/drill-down semantics are losslessly compressed to one cell. Due to its usability and importance, efficient computation of closed cubes still warrants a thorough study.*

In this paper, we propose a new measure, called closedness*, for efficient closed data cubing. We show that* closedness *is an algebraic measure and can be computed efficiently and incrementally. Based on* closedness *measure, we develop an an* aggregation-based *approach, called* C-Cubing *(i.e.,* Closed-Cubing*), and integrate it into two successful iceberg cubing algorithms:* MM-Cubing *and* Star-Cubing*. Our performance study shows that* C-Cubing *runs almost one order of magnitude faster than the previous approaches. We further study how the performance of the alternative algorithms of* C-Cubing *varies w.r.t the properties of the data sets.*

1 Introduction

In this paper, we study the problem of efficiently computating *closed iceberg cubes*. Before formally defining the problem, we first introduce the following definitions.

Definition 1 (*Closed Iceberg Cell*) *In an n-dimension data cube, a cell* $c = (a_1, a_2, \ldots, a_n : m)$ (*where* m *is a measure*) *is called a* k-dimensional group-by cell (*i.e., a cell in a k-dimensional cuboid*)*, if and only if there are exactly* k $(k \leq n)$ *values among* $\{a_1, a_2, \ldots, a_n\}$ *which are* not $*$ (*i.e., all*)*. We further denote* $M(c) = m$ *and* $V(c) = (a_1, a_2, \ldots, a_n)$.

Given a threshold constraint on the measure, a cell is called iceberg cell *if it satisfies the constraint. A popular iceberg constraint on measure count is* $M(c) \geq$ min_sup*, where* min_sup *is a user-given threshold.*

Given two cells $c = (a_1, a_2, \ldots, a_n : m)$ *and* $c' = (a'_1, a'_2, \ldots, a'_n : m')$*, we denote* $V(c) \leq V(c')$ *if for each* a_i $(i = 1, \ldots, n)$ *which is not* $*$*,* $a'_i = a_i$*. A cell* c *is said to be* covered *by another cell* c' *if* $\forall c''$ *such that* $V(c) \leq V(c'') \leq V(c')$*,* $M(c'') = M(c')$*. A cell is called a* closed cell *if it is not covered by any other cells.*

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

Table 1. Example of Closed Iceberg Cube

Given the above definitions, the problem of closed iceberg computation is to *compute all closed cells which satisfy the iceberg constraints*. The closed cube has been shown as a lossless and effective compression of the original data cube in [5, 6]. An example of the closed iceberg cube is given in Example 1.

Example 1 (*Closed Iceberg Cube*) *Table 1 shows a table (with four attributes) in relational database. Let the measure be count, and the iceberg constraint be count* ≥ 2*. Then* $cell_1 = (a_1, b_1, c_1, * : 2)$*, and* $cell_2 = (a_1, *, *, * : 3)$ *are closed iceberg cells; but* $cell_3 = (a_1, *, c_1, * : 2)$ *and* $cell_4 = (a_1, b_2, c_2, d_2 : 1)$ *are not, because the former is covered by* $cell_1$*, whereas the latter does not satisfy the iceberg constraint.* ∎

* The work was supported in part by the U.S. National Science Foundation NSF IIS-02-09199/IIS-03-08215. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

†Hongyan Liu was supported by National Natural Science Foundation of China No. 70471006 /70321001

In this paper, we assume that the measure m is *count*, and the iceberg condition is "$count \geq min\_sup$". Lemma 1 shows that *count* is the fundamental measure comparing with others.

Lemma 1 *If a cell is not closed on measure* count*, it cannot be closed w.r.t. any other measures.*

Rationale. If a cell c_1 is not closed, there must be another cell c_2 which covers c_1. Since both c_1 and c_2 are aggregated by the same group of tuples, they will get the same value for any other measures. One can easily verify that if the measure m is strictly monotonic or anti-monotonic, then closedness on measure *count* is equivalent to measure m. ∎

In closed cubing or mining there are two frequently used operations, *closedness checking* and *pruning non-closed cells/itemsets*; for convenience, we call the former ***c-checking*** and the latter ***c-pruning***. Previous studies on closed cubing or closed frequent pattern mining have developed two major approaches, *output-based c-checking* vs. *tuple-based c-checking*. The former checks the closedness of the result by comparing with previous outputs, while the latter scans the tuples in the data partition to collect the closedness information. Both methods introduce considerable overheads for c-checking (See Section 2 for detailed descriptions). In this paper, we propose a new c-checking method, C-Cubing. With a very small overhead, the closedness information of a cell can be aggregated as a measure, and the c-checking can be simply done by testing this special measure. We call our approach *aggregation-based c-checking*. The new method C-Cubing is implemented in two successful iceberg cubing algorithms: MM-Cubing and Star-Cubing.

The method developed in this paper can be either used for *off-line cube computations* or applied on *online grouping sets (multiple group by) queries*. The contributions of our work are summarized as follows.

- We propose a new measure, called *closedness*, for c-checking. We show that *closedness* is an algebraic measure and can be computed efficiently and incrementally. (Section 3.2)
- We develop a new method, C-Cubing, for the closed cube computation. Our performance study shows that C-Cubing runs almost one order of magnitude faster than previous approaches. We also propose an extended version of Star-Cubing for efficient cube computation in sparse data sets. (Section 3.3,4)
- We make a thorough comparison between iceberg-pruning and c-pruning. A new property of data set, data dependence, is modeled in the synthetic data generation. We believe it will be useful for future algorithm benchmarking. (Section 5.3)

The remaining of the paper is organized as follows. Section 2 discusses the related works. In Section 3, we introduce the *aggregation-based c-checking* approach and integrate it into MM-Cubing. Section 4 proposes an extended version of Star-Cubing and exploits additional c-pruning techniques. A performance study on synthetic and real data sets is reported in Section 5. We conclude our study in Section 6.

2 Related Work

In this section, we discuss the related work in terms of the size complexity of the closed cube and the popular cubing algorithms.

2.1 Size Complexity of Closed Cube

It is well-known that the size complexity (*i.e.*,number of output tuples) of a data cube suffers from the *curse of dimensionality*. Previous work for losslessly compressing cube size includes condensed cubes[10], Dwarf[8, 9] and Qucient Cube[5, 6]. Based on Dwarf[8] model, [9] presents that the size complexity of a uniform coalesced cube is $\mathcal{O}(T^{1+\log_C^D})$, where T is the number of tuples in the base table, D is the number of dimensions, and C is the cardinality of each dimensions. Since generally C is larger than D, the author claims the size complexity of the Dwarf cube does not grow exponentially with the number of dimensions [9].

The compression of Dwarf cube comes from exploiting suffix coalesce[8] in the structure of cube. The suffix coalesce is similar to c-checking. Different from closed cube, the suffix coalesce in Dwarf only checks cells' closedness on the current collapsed dimension, while closed cube extends the c-checking on all previously collapsed dimensions. Hence the number of output cells in Dwarf cube is always no less than that in closed cube.

2.2 Cubing Algorithms

While the size complexity is a major issue of the data cube, to efficiently compute the data cube is another important research problem. In this subsection, we review popular iceberg cubing and closed cubing related algorithms.

BUC [1] first proposes the concept of iceberg cube and employs a bottom-up computation order to facil-

itate the Apriori-based pruning. Star-Cubing [12] exploits more opportunities in shared computation, and uses star-tree structure to integrate simultaneous aggregation into iceberg pruning. MM-Cubing [7] avoids the expensive tree operations in Star-Cubing by partitioning the data into different subspace and using multi-way array aggregation [15](for simplicity, we call this method as MultiWay hereafter) to achieve shared computation.

The concept of *closed cell* is identical to the *upper bound cell* in Quotient Cube [5, 6]. Quotient Cube finds all upper bound cells using a depth-first search algorithm (we refer it as QC-DFS) similar to BUC [1]. To ensure that each output cell is an upper bound, it scans all the dimensions which are not within the group-by conditions, in the current data set partition. This incurs a considerable overhead. The c-checking method in most closed itemsets mining algorithms is to build an external checking architecture, such as a tree or hash-table, which is used to check closedness of later outputs. This method is not suitable for cubing because the size of a data cube, even for the closed one, could be possibly much larger than the original data set. It is undesirable to maintain all the closed outputs in memory.

In summary, we believe that the current closed cubing algorithms are unsatisfactory. It is necessary to exploit more efficient methods. In this paper, we propose an aggregation-based c-checking method, called C-Cubing, and integrate it with MM-Cubing and Star-Cubing.

3 Closed MM-Cubing

MM-Cubing is one of the most adaptive iceberg cubing algorithm. Based on the recent report [7], it outperforms previous algorithms in most cases. Thus, we first develop C-Cubing(MM) by adapting MM-Cubing to closed cube computation.

In this section, we first analyze the challenges of c-checking in MM-Cubing, then introduce a closedness measure to solve the problem, and finally discuss the implementation of C-Cubing(MM) and its efficiency.

3.1 Shared Computation vs. C-Checking

In order to prune the non-closed cells, we need to check the actual tuple values on the $*$ dimensions (dimensions with $*$ as their values) of that cell. Since all $*$ dimensions eventually come out from the dense space (recursive call on sparse spaces does not produce $*$ dimensions), we only need to check it inside the dense subspace. Unlike QC-DFS [6] where the original tuples can be traced and scanned for c-checking, there are no such tuple lists available for each cell in MM-Cubing. Because MM-Cubing uses MultiWay [15] inside the dense space, keeping the tuple ID list for each cell not only introduces computational overhead, but also has storage challenges. Thus, the problem turns out to be how to do c-checking in MultiWay.

The major advantage of MultiWay in cube computation is that MultiWay adopts a simultaneous aggregation approach to share computation among different cuboids. Since we do not keep tuple ID list as in BUC, it is impossible to do c-check in the last moment just before outputting the cell. Based on the computation order of MultiWay, naturally we ask: *Is it possible to compute the closedness information along with the support aggregation?* Given the multi-dimensional aggregation order, this question is equivalent to: *Instead of computing the closedness with a list of tuple IDs, is it possible to compute it incrementally by keeping a summary at each cell?* The answer to this question is yes, and the summary turns out to be a *closedness measure*.

3.2 Closedness Measure

A closedness measure is a value which indicates whether the cell is closed. Before getting into the details of the algorithm, let us first have a look at what kind of measures can be computed incrementally.

Definition 2 *(Distributive Measure) A measure is called* distributive*, if the measure of the whole data set can be computed solely based on the measures of the parts of that data set.*

Definition 3 *(Algebraic Measure) A measure is called* algebraic*, if the measure can be computed based on a bounded number of measures of the parts of that data set.*

Example 2 *It is easy to see that min, count, and sum are distributive measures, e.g., $count(A \cup B) = count(A) + count(B)$, in which A and B denote different parts of the data set; while avg is an algebraic measure since $avg(A \cup B) = (sum(A) + sum(B))/(count(A) + count(B))$.*

Is *closedness* a distributive or an algebraic measure? First, it is not distributive. This can be shown with a simple example: In order to check the closedness of cell $(*, *, 1)$ (here we omit the measure), we check the closedness of $(*, 1, 1)$ and $(*, 2, 1)$. Suppose both $(*, 1, 1)$ and $(*, 2, 1)$ are not closed. It may have several cases. One case is that we have two tuples $(1, 1, 1)$

and $(2, 2, 1)$. In this case, $(*, *, 1)$ is closed. Another case is that we have two tuples, $(1, 1, 1)$ and $(1, 2, 1)$, which means $(*, *, 1)$ is not closed. Thus, it is impossible to derive the closedness of a cell based on the closedness of the subcells. Second, closedness is an algebraic measure because it can be computed based on a distributed measure *Representative Tuple ID* and an algebraic measure *Closed Mask*.

Definition 4 *(Representative Tuple ID) The* Representative Tuple ID *of a cell is the smallest ID of the tuples that aggregate to this cell. In the case the cell is empty (does not contain any tuple), the Representative Tuple ID is set to a special value NULL.*

Lemma 2 *Representative Tuple ID is a distributive measure.*

Rationale. In fact, Representative Tuple ID is the minimum in the set of all related Tuple IDs. We already know that *min* is a distributive measure. Thus, Representative Tuple ID is a distributive measure. ∎

Definition 5 *(Closed Mask) The* Closed Mask *of a cell contains D bits, where D is the number of dimensions in the original database. The bit is 1 if and only if all the tuples aggregated to that cell have the same value in the corresponding dimension.*

Lemma 3 *Closed Mask is an algebraic measure.*

Proof. We show how to use the Closed Masks and the Representative Tuple IDs of subsets to compute the Closed Mask for the whole set.

We denote the subsets by $S_i,\ i \in \{1, 2, \ldots, k\}$, and the whole set by S, clearly, $S = \bigcup_{i=1}^{k} S_i$. Let $C(S_i, d)$ be the Closed Mask of set S_i on dimension d, $T(S_i)$ be the Representative Tuple ID of set S_i, and $V(t, d)$ be the value of Tuple t on dimension d.

By the definition of Representative Tuple ID, we have $T(S) = min_{i=1}^{k} T(S_i)$. From the definition of Closed Mask, the bit is 1 if and only if all the tuples belonging to that cell have the same value in the corresponding dimension, in which case the following two conditions hold: (1) all the Closed Masks of the subsets should have 1 on that bit, and (2) all the Representative Tuples of the subsets have the same value on that dimension. This leads to the following equation:

$$C(S, d) = \prod_{i=1}^{k} C(S_i, d) \times Eq(|\{V(T(S_i), d),\ 1 \le i \le k\}|, 1)$$

where $|\{V(T(S_i), d),\ 1 \le i \le k\}|$ means the number of distinct values in the set $\{V(T(S_i), d),\ 1 \le i \le k\}$, and $Eq(x, y)$ is 1 if x is equal to y, otherwise it evaluates to 0. The base case for $C(S, d)$ is that the set S only contains one tuple, at that time, the value of $C(S, d)$ is 1 for all d.

This equation defines how to compute the Closed Mask incrementally based on Closed Masks and Representative Tuple IDs of the subsets. Thus, Closed Mask is an algebraic measure. ∎

Please note in all these equations, multiplication can be replaced by a bitwise-and operation, which is more efficient. We may also notice that Representative Tuple ID can be the ID of any tuple related to the cell. We use the minimum intentionally to ease problem formulation and discussion.

Now we are able to define the closedness measure.

Definition 6 *(All Mask) The* All Mask *of a cell is a measure consisting of D bits, where D is the number of dimensions. The bit is 1 if and only if that cell has a value of $*$ in the corresponding dimension.*

Definition 7 *(Closedness Measure) Given a cell, whose closed mask is C and all mask is A, the* closedness measure *is defined as C&A, where & is bitwise-and operations.*

Example 3 *The All Mask of a cell $(*, *, 2, *, 1)$ is $(1, 1, 0, 1, 0)$. Please note All Mask is a property of the cell and can be computed directly. If the closed mask of this cell is $(1, 0, 1, 0, 0)$, then its closedness measure value is $(1, 0, 0, 0, 0)$.*

Since the *all mask* can be directly computed from the cell and the *closed mask* is an *algebraic measure*, we conclude that the *closedness* is also an *algebraic measure*.

Lemma 4 *Closedness is an algebraic measure.*

A cell is non-closed if and only if in at least one dimension of its closedness measure, the bit is 1. Otherwise, the cell is closed. The intuitive explanation is that the cell is non-closed if and only if (1) all tuples of the cell have the same value in a dimension, and (2) the value of the cell in that dimension is $*$.

Since closedness of a cell is an algebraic measure, we can compute it incrementally in the simultaneous aggregation steps of the dense subspace. Before the output step, we need to check the closedness of the cell. We will output the cell only if it is closed.

3.3 Implementation of C-Cubing(MM)

We have implemented the closedness measure in MM-Cubing. The new algorithm is called C-Cubing(MM). The integration of closedness measure

into MM-Cubing is as follows. Whenever there is an aggregation of count, we aggregate closedness measure as well. When a cell is ready to output, the closedness measure is checked. Since we do not do any less computation (though sometimes we do not output the cell, which means less I/O operations), we can expect that C-Cubing(MM) always has overheads over MM-Cubing.

However, we can also expect that C-Cubing(MM) will not degrade too much from MM-Cubing. For space efficiency, the additional data structures used are Closed Mask and Representative Tuple ID. The sizes of the Close Mask and the Representative Tuple ID are both proportional to the size of the aggregation table, which is generally limited to 4MB[7]. For time efficiency, the Closed Mask and Representative Tuple ID are both aggregated in the same way as the support. Most of time, these aggregations only involve bits operations. Thus, the additional cost is cheap w.r.t. the existing cost of aggregation. That is, these modifications do not introduce large space or time overheads.

The technique we use in C-Cubing(MM) is only related to c-chcking, that is, we check the closedness of a cell just before the output step. Correspondingly, when a new partition (or child tree in Star-Cubing) is to be computed, if we can identify in advance that all the cells which will be generated by this partition are not closed, then the whole partition can be simply bypassed. This is related to c-pruning. Obviously, c-pruning is more promising. We will discuss how to achieve c-pruning with Star-Cubing in the next section.

4 Closed StarCubing

We select Star-Cubing for closed cubing study because we believe tree-like data structure is useful for closed pattern mining, since data are no longer individual tuples, but are organized by trees, and their relationships can be partially implied by the tree structure. In this section, we discuss closed iceberg cubing algorithm derived from Star-Cubing.

As seen in Section 2, Star-Cubing is inefficient in sparse data set due to the high cost in the tree operations. To lift this restriction, we extend the original Star-Cubing algorithm in two ways: First, we use a new data structure called *StarArray* to represent a cuboid tree; second, we exploit a new computational order to generate child trees. C-pruning and c-checking methods are further applied on both the original Star-Cubing and extended StarArray algorithms.

Before we discuss the detail of the algorithm, it is necessary to first clarify the related terminology. In this paper, we use *father* and *son* when we refer to *node*, *parent* and *child* when we talk about *tree*. A tree is called *base tree* if it contains all tuples with full dimensions.

4.1 StarArray

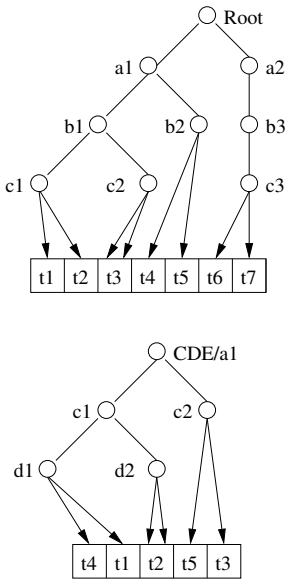
Star-Cubing algorithm uses the star tree to completely represent the whole data set. However, it is quite clear that only upper-level nodes in the tree gain from the shared aggregation. To avoid high maintenance cost in the lower level part, we introduce a hybrid structure, StarArray. A StarArray is constituted by a couple $\langle A, T\rangle$, where A is an array storing tuple IDs, and T is a partial tree whose leaf nodes pointing to a continuous subpool of A. Different from Star-Cubing where the nodes are constructed until the end of the dimension is reached, in StarArray, when we find a node whose aggregation value is less than *min_sup*, all the subbranches below this node are truncated. Instead, the corresponding tuple IDs are copied into A, and the leave node will have two pointers pointing to the start and end positions in A.

Example 4 *(StarArray) Given a data set (as in Fig. 1) and* $\min\_\sup(M = 3)$ *for iceberg cubing, the base StarArray and a child StarArray* CDE/a_1 *created from node* a_1 *are shown in Fig. 1, each leaf node points to a sub pool of the array. The tuple IDs in the array are partially ordered according to the remaining dimensions which were not shown in the tree. For example, the* c_1 *node in the base tree points to* t_1, t_2*, which are first ordered by dimension D, then ordered by dimension E. Similarly, for the* d_1 *node in* CDE/a_1 *tree,* t_4 *is placed before* t_1 *because of the ordering in dimension E.*

One of the major benefit of Star-Cubing is using Simultaneous Aggregation, that is, the parent tree only need to be traversed once to create all child trees. However, there is a hidden overhead on traversing child trees multiple times in the process of its creation. With StarArray, we use a new method called Simultaneous Traversal, where the parent tree is traversed multiple times(one time for each child tree), but each child tree will be only traversed once during the process of creation. It is provlable that in sparse data, the Simultaneous Traversal method is more efficient than Simultaneous Aggregation. Due to the limited space, we omit the details in this paper. The complete description can be found in the extended version[13].

4.2 C-Pruning by Closedness Measure

In Section 3, we have seen the closedness measure can be efficiently integrated into a previous algorithm



tid	A	B	C	D	E
t1	a1	b1	c1	d1	e2
t2	a1	b1	c1	d2	e2
t3	a1	b1	c2	d2	e1
t4	a1	b2	c1	d1	e1
t5	a1	b2	c2	d1	e1
t6	a2	b3	c3	d1	e1
t7	a2	b3	c3	d2	e1

Figure 1. StarArray: The base tree and a child tree

to get a new algorithm for closed iceberg cube computation(we call it closed version of MM-Cubing). Using the similar technique, we can aggregate closedness measure in Star-Cubing. In this section, we first discuss how closedness measures are computed, and then exploit more efficient methods by c-pruning.

Star-Cubing uses a tree structure where each node keeps the count measure, which corresponds to the set of aggregated tuples. We add the closedness measure in each node. Initially, every node in the base tree holds a closedness measure, which includes a couple *Closed Mask* and *Representative Tuple ID*. Given a dimension order, each level of the tree corresponds to one specific dimension. Assume the node under investigation is on level i (*e.g.*, belong to dimension i), then the Closed Masks from 1 to i bits are set as "1", leaving the others as "0". For example, in Fig. 1, node c_1 in the base tree has Closed Mask $(1, 1, 1, 0, 0)$, which means all the tuples aggregated to the node c_1 (*i.e.*, tuples t_1 and t_2) share the same value on the first three dimensions. The representative tuple ID of c_1 is chosen as t_1. Note the Closed Masks in nodes are partial in the way that they only keep the closed information of the former dimensions, we adopt this approach for three reasons. First, the Star-Cubing algorithm only outputs the cells at the last two levels, where the Closed Masks are either complete (*i.e.*, at the leaves) or easy to extend to a complete one (*i.e.*, the last second level). Internal nodes will never output a cell. Thus it is safe to allow a partial closed information. Second, the closed information on the former dimensions can be got without additional work (just following the tree structure), while the closed information on the latter dimensions needs a tree traversal, which clearly has considerable overhead. Finally and most importantly, partial closed information is already enough for c-pruning.

We show how the closedness measure is aggregated on trees. Similar to the *All Mask* in Section 3.2, a *Tree Mask* is assigned on each tree. The tree mask contains D bits, where D is the number of dimensions. The tree mask of the base tree has "0" on every bit. Whenever there is a child tree created, it first inherits the tree mask of the parent tree, then switches the bit on the collapsed dimension, from "0" to "1". For example, in Fig. 1, the CDE/a_1 child tree has tree mask $(0, 1, 0, 0, 0)$, since the dimension B was collapsed. Assuming node N is aggregated by nodes $N_i, i \in \{1, 2, \ldots, k\}$, we use the same notations as in lemma 3, that is, $C(N_i, d)$ is the closed mask on dimension d of node N_i, $T(N_i)$ is the representative tuple ID of node N_i, $V(t, d)$ is the value of tuple t on dimension d, and $Eq(x, y) = 1$, if $x = y$; otherwise, it is 0. We further define $TM(d)$ is the value of tree mask on dimension d. We have:

$$C(N,d) = \begin{cases} \prod_{i=1}^{k} C(N_i, d) & if \;\; TM(d) = 0 \\ \prod_{i=1}^{k} C(N_i, d) \times Eq(|\{V(T(N_i, d), & \\ \quad 1 \le i \le k\}|, 1) & otherwise \end{cases}$$

The above equation means: If $TM(d) = 0$, then dimension d is not a (previously) collapsed dimension. We should reserve the partial closed mask to be consistent with tree structure. If $TM(d) = 1$, and further if the Closed Masks of all nodes agree on dimension d where they all have value "1", we check the number of distinct values on dimension d. If it is larger than 1, the corresponding bit is set to "0"; otherwise, the bit will be set as "1". As an example, the Closed Masks of node c_1 and d_2 of child tree CDE/a_1 are $(1, 0, 0, 0, 0)$ and $(1, 1, 1, 0, 0)$, respectively.

Based on the Closed Mask and Tree Mask, we have developed two lemmas to facilitate the c-pruning.

Lemma 5 *(C-Pruning by Closed Mask) In closed cube computation, assuming the Closed Mask of a node is C, and the Tree Mask is TM, if $C\&TM \neq 0$ (where & is a bitwise-and operation), then all the cells which are output by the branches under this node or by the child trees created from the branches under this node are non-closed. Particularly, if the node is at the last level of the tree, the output is pruned.*

Rationale. If $C\&TM \neq 0$, then there is at least one previously collapsed dimension (say d), on which all the tuples aggregated to the current node share the same

value v_d. Since this dimension has been collapsed, all the cells that will be output by the branch under the current node or by child trees created from this branch (including current node) have value $*$ on the dimension d. However, for each such cell, there is at least a cell (*i.e.*, the cell whose value on the dimension d is v_d) which covers it. That is, all the cells are not closed. Hence, it is safe to prune the unnecessary computation on the whole partition. Particularly, if the node is at the last level of the tree, then the output cell is not closed. ∎

The second pruning rule is about a special subtree structure, *Single Path*, which is a branch of a tree where each node in the path has only one son.

Example 5 *(Single Path) In Fig. 1, a_2b_3 is a single path. c_3 does not belong to the single path, since c_3 has two sons.*

Lemma 6 *In closed cube computation, if a node belongs to a single path, then all the cells which are output by the child tree created from this node are non-closed. Particularly, if the node is at the last second level, then the output is pruned.*

Rationale. New child tree created from a node F is obtained by collapsing its son nodes. If a node belongs to a single path, then this node has only one son S. Assuming the corresponding dimension is d_s, the tree mask of the new child tree on bit d_s will be set as "1". In the meantime, all the nodes in the new child will share the same value on dimension d_s, that is, the bit d_s of their closed mask will be set as "1", according to Lemma 5, all the nodes in the child tree will be pruned. ∎

Since we do c-pruning on all output levels, the c-checking is complete. The possible efficiency comes from the c-pruning of internal nodes. We have applied the above c-pruning method to both the original and the extended Star-Cubing. The two new algorithms are named as C-Cubing(Star) and C-Cubing (StarArray), respectively.

5 Performance Study

To check the efficiency and scalability of the proposed algorithm, a comprehensive performance study is conducted by testing our implementations of closed iceberg cubing. We compare the new methods with QC-DFS, which, to our knowledge, is the best available method on closed cubing. C-Cubing(MM), C-Cubing(Star), and C-Cubing (StarArray) are all coded using C++, and the QC-DFS is provided by the author of [6]. The experiments is carried out on an Intel Pentium-4 3.2GHz system with 1G of RAM running windows XP. The time recorded includes both the computation time and the I/O time. Similar to other performance studies in cube computation [15, 1, 4, 12, 7], we assume that all the data sets used in experiments could fit in main memory, and we will discuss the scalability issue in section 6.

Experiments are conducted on both synthetic and real datasets. For synthetic data, we use following terms to describe the dataset: $\mathcal{D}$ the number of dimensions, $\mathcal{C}$ the cardinality of each dimension, $\mathcal{T}$ the number of tuples in the base cuboid, $\mathcal{M}$ the minimum support level, and $\mathcal{S}$ the skewness or `zipf` of the data. When $\mathcal{S}$ equals 0.0, the data is uniform. As $\mathcal{S}$ increases, the data is more skewed. $\mathcal{S}$ is applied to all the dimensions in a particular data set.

For real dataset, we use the weather dataset SEP83L.DAT in [3], which has 1,002,752 tuples with selected 8 dimensions. The attributes (cardinalities) are as follows: year month day hour (238), latitude (5260), longitude (6187), station number (6515), present weather(100), change code (110), solar altitude (1535) and relative lunar illuminance (155).

5.1 Computing Full Closed Cube

The first set of experiments compare four algorithms, *i.e.*, C-Cubing(MM), C-Cubing (StarArray), C-Cubing(Star) and QC-DFS, for full closed cube computation (*i.e.*, *min_sup* = 1). The performance of the four algorithms are compared with respect to tuple size (Fig. 2), cardinality (Fig. 3), skewness (Fig. 4) and dimension (Fig. 5) using synthetic datasets and the weather data set. For simplicity, we abbreviate C-Cubing to *CC*.

In Fig. 2 , we randomly generate data sets with 10 dimensions, varying the number of tuples from 200K to 1000K. In Fig. 3, we test the performance by increasing the cardinalities of each dimension from 10 to 1000. We further tune the skewness of the data from 0 to 3 in Fig. 4. The tuple size for the latter three datasets is 1000K. Finally, The real dataset, weather data, is tested by selecting the first 5 to 8 dimensions in Fig. 5.

There are three main points that can be drawn from these results. First of all, all the three new algorithms that use *aggregation-based c-checking* are consistently more efficient than QC-DFS, on both synthetic and real datasets. C-Cubing(Star) and C-Cubing (StarArray) achieve better performance comparing with C-Cubing(MM) because the Star family algorithms exploit more c-pruning. Secondly, when the cardinality is relatively low, C-Cubing(Star) performs better than C-Cubing (StarArray). However, when the cardinality

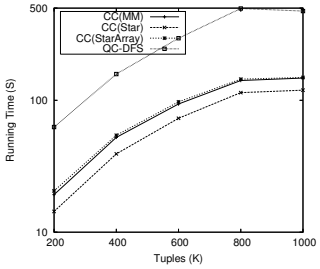


Figure 2. Closed Cube Computation w.r.t. Tuples, where $\mathcal{D} = 10$, $\mathcal{C} = 100$, $\mathcal{S} = 0$, $\mathcal{M} = 1$

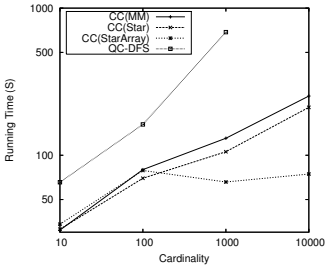


Figure 3. Closed Cube Computation w.r.t. Cardinality, where $\mathcal{T} = 1000K$, $\mathcal{D} = 8$, $\mathcal{S} = 1$, $\mathcal{M} = 1$

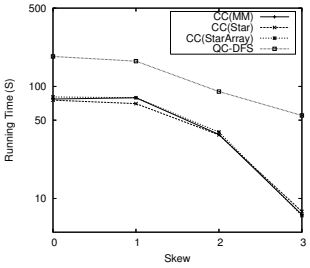


Figure 4. Closed Cube Computation w.r.t. Skew, where $\mathcal{T} = 1000K$, $\mathcal{C} = 100$, $\mathcal{D} = 8$, $\mathcal{M} = 1$

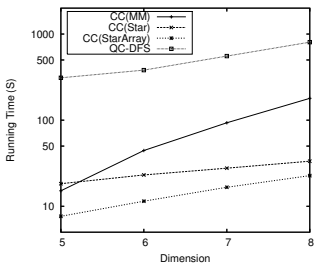


Figure 5. Closed Cube Computation w.r.t. Dimension, Weather Data Set, $\mathcal{M} = 1$

increases, C-Cubing (StarArray) becomes better. This is consistent with our analysis in Section 4. In low cardinality, Simultaneous Aggregation in C-Cubing(Star) is promising, while in high cardinality, Simultaneous Traversal of C-Cubing (StarArray) is better. QC-DFS performs much worse in high cardinality because the counting sort costs[1] more computation. Finally, all three new algorithms get performance improvements as the skewness increases. C-Cubing(MM) benefits from the nonuniform distribution where the dense and sparse spaces are easier to be identified. The two Star family algorithms, C-Cubing(Star) and C-Cubing (StarArray), are faster because the tree structure shares more computation in the dense parts of the data, and c-pruning is more effective in sparse parts.

5.2 Closed Iceberg Cubing

The second set of the experiments compare the performances of closed iceberg cubing (Fig. 6–9). We are not able to compare with QC-DFS in this section since QC-DFS does't implement iceberg cubing in it. Moreover, the previous works [12, 7] already reported that, in iceberg cubing, MM-Cubing and Star-Cubing outperform BUC, from which QC-DFS is derived. The curves in the last subsection also clearly demonstrated our methods are more efficient when taking full closed cubing into consideration.

The performance is compared with respect to *min_sup*, data skew and cardinality in synthetic data. The data set used in Fig. 6 has 1M tuples, 8 dimensions and 0 skewness. The *min_sup* is increased from 2 to 16. The parameters of data sets in Fig. 7 are the same as Fig. 6, except the skewness is varied from 0 to 3. The cardinalities are increased from 10 to 10000 in Fig. 8.

We can see that C-Cubing(MM) performs better when *min_sup* increases. This is because the iceberg pruning gets more effective. The factor of c-pruning is decreasing since some of the closed cells are ruled out by iceberg condition. Fig. 9 uses the same weather data with 8 dimensions. It is observed that the switching point between C-Cubing(MM) and the Star family algorithms in this experiment is higher than those in the synthetic datasets.

Comparing with the results in the full closed cube computation, we have seen a totally different behavior of the algorithms. Generally speaking, Star family algorithms work well while *min_sup* is low. They are outperformed by C-Cubing(MM) as the *min_sup* increases. We are interested in where are the switching points and how the properties of the data set affect the switching points. We conduct another group of experiments for this purpose.

5.3 C-Pruning vs. Iceberg Pruning

In this subsection, we first generalize the observations in the last two sets of experiments, which lead to an important data set property: data dependence. We then show how the data dependence will affect the performance of the algorithms.

Closed cells imply the dependence of the values on different dimensions. For example, if cell $cell_1 = (a_1, b_1, c_1)$ is closed and it covers cell $cell_2 = (a_1, b_1, *)$, then there exists a *dependence rule*: $(a_1, b_1) \rightarrow c_1$. Most data sets in our experiments are very sparse, that is, the feature space size (by multiplying all the cardinalities) is much larger than the number of tuples. Sparse data sets contain lots of trivial closed cells whose support is 1. When *min_sup* is 1, trivial closed cells dominate the outputs. However, when *min_sup* increases, these trivial closed cells are already pruned by iceberg conditions. Generally, the c-pruning is more effective when the dependence in the data is higher, since more closed cells will "survive" from the iceberg

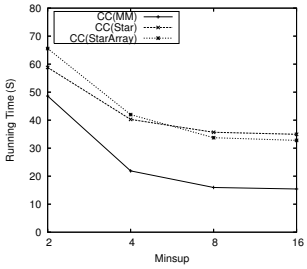


Figure 6. Closed Iceberg Cube Computation w.r.t. Minsup, where $\mathcal{T} = 1000K$, $\mathcal{C} = 100$, $\mathcal{S} = 0$, $\mathcal{D} = 8$

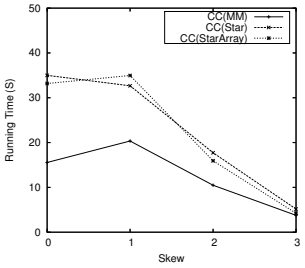


Figure 7. Closed Iceberg Cube Computation w.r.t. Skew, where $\mathcal{T} = 1000k$, $\mathcal{D} = 8$, $\mathcal{C} = 100$, $\mathcal{M} = 10$

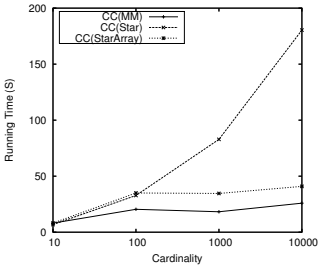


Figure 8. Closed Iceberg Cube Computation w.r.t. Cardinality, where $\mathcal{T} = 1000K$, $\mathcal{D} = 8$, $\mathcal{S} = 1$, $\mathcal{M} = 10$

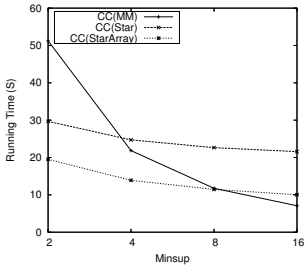


Figure 9. Closed Iceberg Cube Computation w.r.t. Minsup, Weather Data Set, $\mathcal{D} = 8$

pruning.

To verify this hypothesis, we incorporate data dependence into our synthetic data generation. The data dependence is modeled as *dependence rules*. An example of this kind of rules is as discussed above: $(a_1, b_1) \rightarrow c_1$, which means that if a_1 appears on dimension A, and b_1 appears on dimension B, then the value on dimension C will be fixed to c_1. This kind of rules exist widely in real databases. For example, in weather data, when a certain weather condition (a_1) appears at a specific time (b_1) of a day, there is always a unique value for solar altitude (c_1).

Each rule corresponds to a *pruning power*, which means how many cells will be pruned by this rule. Assuming the original cube size is S, the portion which contains a_1 and b_1 has an estimated size of $\frac{S}{Card(A) \times Card(B)}$. Further dividing the portion by the value on dimension C results in $Card(C) + 1$ classes, which have the form of $(a_1, b_1, *, \ldots)$, $(a_1, b_1, c_1, \ldots)$, $\ldots$, $(a_1, b_1, c_{Card(C)}, \ldots)$. After applying the rule $(a_1, b_1) \rightarrow c_1$, only cells $(a_1, b_1, c_1, \ldots)$ will be left. Thus the pruning power of this rule is estimated by $\frac{Card(C)}{Card(A) \times Card(B) \times (Card(C)+1)}$. The pruning power of a set of rules is evaluated by accumulating their individual pruning power. Then the data dependence R is measured as $R = -\sum_{i=1}^{n} \log(1 - pruning\ power(rule_i))$. The larger the value of R is, the more dependent is the dataset.

To compare the c-pruning and iceberg pruning, we run a set of experiments by varying data dependence R and *min_sup*. Fig. 10 shows the performances of C-Cubing(MM) and C-Cubing(Star) with respect to the value of R. The data has 400K tuples with 8 dimensions, each of which has cardinality 20. The *min_sup* is set to 8. Fig. 11 shows a more comprehensive study on the algorithm performance w.r.t. R and *min_sup*, while the other parameters of the data are kept the same as above. The value of R is varied from 1 to 3, while the value of *min_sup* is increased from 1 to 512. For a given parameter set, the ID of the best algorithm is plotted. Since the test data sets have relatively low cardinalities, C-Cubing(Star) performs better than C-Cubing (StarArray). We can expect that if the cardinality is large, C-Cubing (StarArray) will be better.

It is clear that the performance of C-Cubing(MM) and the Star family algorithms are correlated to the extent that c-pruning can be achieved w.r.t. the iceberg pruning. The Star family algorithms are promising when the c-pruning is comparatively large, and C-Cubing(MM) is better when the iceberg pruning dominates.

5.4 Overhead of c-checking

In this section, we examine the overhead of our c-checking method. There are additional costs to aggregate the closedness measure, but we will show that it is not a major expense in the context of computing the closed iceberg cube. Furthermore, the c-pruning in C-Cubing(Star) and C-Cubing (StarArray) actually reduces the overall computational cost.

We compare C-Cubing(MM) and C-Cubing (StarArray) with MM-Cubing and the non-closed version of StarArray, respectively. The output is disabled to eliminate the difference introduced by I/O. We tested all the algorithms on the weather data. Fig. 12 shows the comparison on C-Cubing(MM) and MM-Cubing. To our surprise, C-Cubing(MM) performs better than MM-Cubing in low *min_sup*. This is because a simple optimization is exploited in C-Cubing(MM): When a subspace whose size is equal to the *min_sup* is found, C-Cubing(MM) will directly output the closed cell, while MM-Cubing will enumerate all the combinations (though they are not output). When the *min_sup* is high, MM-Cubing performs better, but the overhead of

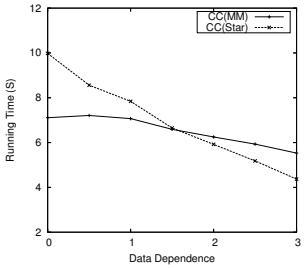


Figure 10. Cube Computation w.r.t. Data Dependence, where $\mathcal{T} = 400K, \mathcal{D} = 8,\ \mathcal{C} = 20,\ \mathcal{S} = 0,\ \mathcal{M} = 16$

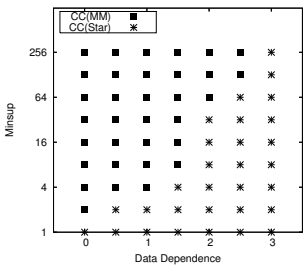


Figure 11. Best Algorithm, Varying Minsup and Dependence, where $\mathcal{T} = 400k,\ \mathcal{D} = 8,\ \mathcal{C} = 20,\ \mathcal{S} = 0$

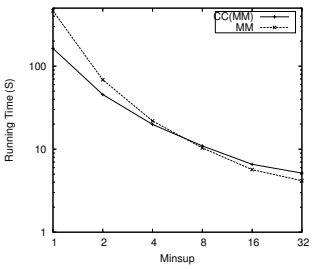


Figure 12. Overhead of c-checking (MM-Close) w.r.t. Minsup, Weather Data Set, where $\mathcal{D} = 8$

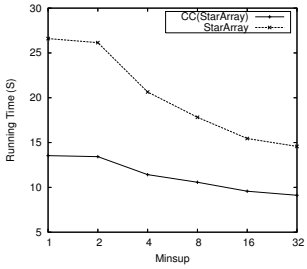


Figure 13. Benefits of C-Pruning (StarArray) w.r.t. Minsup, Weather Data Set, Where $\mathcal{D} = 8$

C-Cubing(MM) is within 10% of the total time.

Fig. 13 shows the performance of C-Cubing (StarArray), the closed version runs faster than the non-closed version, especially when *min_sup* is low, since at that time the c-pruning is more effective.

In summary, we have systematically tested three closed iceberg cubing algorithms: C-Cubing(MM), C-Cubing(Star), and C-Cubing (StarArray), with the variations of cardinality, skew, *min_sup*, and data dependence. The Star family algorithms perform better when *min_sup* is low. C-Cubing(MM) is good when *min_sup* is high. The switching point of *min_sup* increases with the dependence in the data. High dependence incurs more c-pruning, thus it benefits the Star algorithms. Comparing C-Cubing(Star) and C-Cubing (StarArray), the former is better if the cardinality is low; otherwise, C-Cubing (StarArray) is better.

6 Conclusions

For efficient computation of closed (iceberg) cubes, we have proposed an aggregation-based c-checking approach, C-Cubing. With this approach, we proposed and implemented three algorithms: C-Cubing(MM), C-Cubing(Star) and C-Cubing (StarArray). All the three algorithms outperform the previous approach. Among them, we have found C-Cubing(MM) is good when iceberg pruning dominates the computation, whereas the Star family algorithms perform better when c-pruning is significant.

Further compression of closed cubes and extension of our method to computing closed frequent structured patterns are two interesting issues for future research.

References

[1] K. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg Cubes. SIGMOD'99.

[2] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Data Mining and Knowledge Discovery, 1, 1997.

[3] C.J. Hahn et al. 1994: Edited Synoptic Cloud Reports from Ships and Land Stations Over the Globe, 1982-1991. (http://cdiac.ornl.gov/ftp/ndp026b/)

[4] J. Han et al. Efficient Computation of Iceberg Cubes with Complex Measures. SIGMOD'01.

[5] L. Lakshmanan et al. Quotient Cubes: How to Summarize the Semantics of a Data Cube. VLDB'02.

[6] L. Lakshmanan et al. QC-Trees: An Efficient Summary Structure for Semantic OLAP. SIGMOD'03.

[7] Z. Shao et al. MM-Cubing: Computing Iceberg Cubes by Factorizing the Lattice Space. SSDBM'04

[8] Y. Sismanis et al. Dwarf: Shrinking the PetaCube. SIGMOD'02.

[9] Y. Sismanis et al. The Polynomial Complexity of Fully Materialized Coalesced Cubes. VLDB'04.

[10] W. Wang et al. Condensed Cube: An Effective Approach to Reducing Data Cube Size. ICDE'02.

[11] J. Wang et al. Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. KDD'03.

[12] D. Xin et al. Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration. VLDB'03.

[13] D. Xin et al. C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking. Technical Report UIUCDCS-R-2005-2648, Department of Computer Science, UIUC, October 2005.

[14] M. Zaki and C. Hsiao. Charm: An Effcient Algorithm for Closed Association Rule Mining. SDM'02.

[15] Y. Zhao et al. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. SIGMOD'97.