

## **MDAG-Cubing: A Reduced Star-Cubing Approach**

**Joubert de Castro Lima<sup>1</sup>, Celso Massaki Hirata<sup>1</sup>**

<sup>1</sup>Instituto Tecnológico de Aeronáutica (ITA) – Divisão de Ciência da Computação  
Praça Marechal Eduardo Gomes, 50 - Vila das Acácias – 12.228-900 – São José dos  
Campos – SP – Brazil

{joubert, hirata}@ita.br

***Abstract.** In this paper, we extend the Star-Cubing approach by introducing a new hybrid dimension-based approach to efficiently compute full or iceberg cubes with simple or complex measures. This new approach, named Multidimensional Direct Acyclic Graph Cubing (MDAG-Cubing), introduces the notion of external and internal nodes to reduce the cube representation without loss of generality. The reduced representation enables improvements in memory consumption and data structure traversals, since it has smaller height, fewer nodes and fewer branches. We implement two MDAG algorithms that run, on average, 25-50% faster than the Star-Cubing algorithm and consume 70-90% less memory to represent the same data cube.*

### **1. Introduction**

Since the introduction of Data Warehouse and OLAP [Gray et al. 1997], efficient computation of data cubes has become one of the most relevant and pervasive problems in research. This problem is of exponential complexity with respect to the number of dimensions; therefore, the materialization of a cube involves a huge number of cells and also a substantial amount of time for its generation.

In [Xin et al. 2007], the previous cube computation studies were classified into five categories: (1) efficient computation of full or iceberg cubes with simple or complex measures [Zhao et al. 1997], [Beyer et al. 1999], [Han et al. 2001], [Xin et al. 2007], (2) selective materialization of views [Baralis et al. 1998], [Shukla et al. 1998], (3) computation of compressed data cubes by approximation such as quasi-cubes and wavelet cubes [Barbará et al. 1997], [Vitter et al. 1998], (4) computation of condensed, dwarf, closed or quotient cubes [Wang et al. 2002], [Sismanis et al. 2002], [Xin et al. 2006], [Lakshmanan et al. 2002], and (5) computation of stream cubes for multidimensional analysis [Chen et al. 2002]. Among these categories, it is recognized that the first one, efficient computation of full or iceberg cubes with simple or complex measures, is the basis to all others, and any new approach developed to address it, may strongly influence new developments related to other categories.

The problem of cube computation can be formally defined as follows: given a base relation  $D$  and  $n$  attributes, a cell  $a = (a_1; a_2; \dots; a_n; c)$  in an  $n$ -dimension data cube is a “GROUP BY” with measure  $c$  and  $n$  distinct attribute values  $a_1; a_2; \dots; a_n$ . A cell  $a$  is called an  $m$ -dimensional cell, if and only if there are exactly  $m$  ( $m \leq n$ ) values among  $\{a_1; a_2; \dots; a_n\}$  which are not  $*$  ( $*$  is a wildcard for all values). It is called a base

cell if  $m = n$ . Given a base *cuboid* the task is to compute an iceberg cube, i.e., the set of cells which satisfies an iceberg condition, or the full cube if there is no such condition.

Previous studies have developed three major dimension-based approaches to efficient compute full or iceberg cubes with simple or complex measures: top-down [Zhao et al. 1997], bottom-up [Beyer et al. 1999], [Han et al. 2001] and hybrid [Xin et al. 2007].

The hybrid dimension-based Star-Cubing approach, proposed in [Xin et al. 2003], outperforms the other approaches in dense, skewed and not so sparse relations. However, in very sparse relations, it has to handle the additional traversal cost introduced by the star-tree structure. In [Xin et al. 2007], the authors developed some single tree path aggregation optimizations to turn the Star-Cubing a promising approach even in very sparse relations, but we still have problems caused by:

- (i) In Star-Cubing approach the *all* (\*) value and the infrequent attribute values are replaced by a wildcard to reduce the cube representation. However, this strategy does not eliminate the problem, but just appeases it. In sparse relations the number of wildcards can be large and also the star-tree height can become high with intermediate wildcards, so multiple tree traversals continue costly;
- (ii) The star-tree is a prefixed data structure that produces a huge number of suffixed attributes redundancies. This redundancy not only consumes a huge amount of memory but also requires a substantial amount of time for its generation. The suffixed attributes redundancies also increase the number of unnecessary star-tree branches that must be traversed.

In this paper, we propose a novel reduced approach that extends the Star-Cubing approach in terms of cube representation and computation. This new approach, named Multidimensional Direct Acyclic Graph Cubing (MDAG-Cubing), significantly reduces the costly data structure traversals with the following properties:

- (i) It does not consider wildcards, representing *all* (\*) and infrequent attributes, on its cube representation;
- (ii) It reduces the number of redundant suffixed attributes, the data structure height and the number of data structure branches with the notion of external and internal nodes. In our approach, the external nodes represent, in the worst case, half of the star-tree nodes (not including the wildcards). The remaining star-tree nodes are represented by our internal nodes, so each external node has a set of shared internal nodes, forming a DAG that has fewer nodes, smaller height and fewer branches when compared with the star-tree;
- (iii) It does not generate new internal and external nodes aggregations when the number of descendants is unique or the internal nodes descendants are different. Instead, the MDAG approach shares the existing descendant nodes aggregations among the ancestral nodes.

These properties reduce the MDAG cube computation effort and the MDAG cube representation size. We implement two MDAG algorithms that run, on average,

25-50% faster than the Star-Cubing algorithm and consume 70-90% less memory to represent the same data cube.

The remaining of the paper is organized as follows. Section 2 describes the hybrid dimension-based Star-Cubing approach. In Section 3, the MDAG-Cubing properties are explained and the algorithm is proposed. The performance results are presented in Section 4. Discussions on the potential extensions and limitations of the MDAG-Cubing approach are in Section 5, and we conclude our study in Section 6.

## 2. Hybrid Dimension-Based Star-Cubing Approach

The Star-Cubing approach uses a prefixed generic search tree, named star-tree, to represent individual *cuboids*. Each level in the tree represents a dimension, and each node represents an attribute.

This representation collapses the common prefixes to save memory usage and to allow value aggregation at internal nodes. A new wildcard, called "star", is used in order to represent the infrequent attributes. All infrequent attributes are replaced by *star*, so the iceberg *cuboid* trees can be reduced. The reduction is lossless, based on Apriori property.

The Star-Cubing approach takes the advantages of the two cube computation approaches (top-down and bottom-up). On the global computation order, it uses the simultaneous top-down aggregation, similar to [Zhao et al. 1997]. However, it has a sub-layer based on the bottom-up approach by exploring the notion of shared dimension. This integration allows the Star-Cubing approach to aggregate on multiple dimensions while partitioning parent group-by's and prune child group-by's that do not satisfy the iceberg condition.

In [Xin et al. 2007], the authors extended the original approach by introducing some single tree path aggregation optimizations to save unnecessary traversals and also eliminate redundant star-tree nodes creation. The extended Star-Cubing approach outperforms the previous methods, proposed in [Zhao et al. 1997], [Beyer et al. 1999], [Han et al. 2001] and [Xin et al. 2003], in dense, sparse and skewed scenarios.

The Star-Cubing approach generates, in the worst case,  $\sum_{i=1}^d \prod_{j=1}^i (C_j + 1)$  nodes to represent a cube when the wildcard is considered at leaf nodes or  $\left( \sum_{i=1}^d \prod_{j=1}^i (C_j + 1) \right) - \left( \prod_{i=1}^{d-1} (C_i + 1) \right)$  nodes when it is not considered, where  $C$  is the cardinality of each dimension and  $d$  is the number of dimensions.

## 3. MDAG-Cubing Approach

The MDAG-Cubing approach represents a data cube using a Direct Acyclic Graph (DAG) with multidimensional properties. We use DAG to represent individual *cuboids*. Each level in the DAG represents any dimension, and each node represents an attribute. Tuples in *cuboid* are inserted into the MDAG in the same way they are inserted into the star-tree.

In MDAG-Cubing approach, we introduce the notion of external and internal nodes. We can represent a data cube using only ancestral, sibling and descendant nodes,

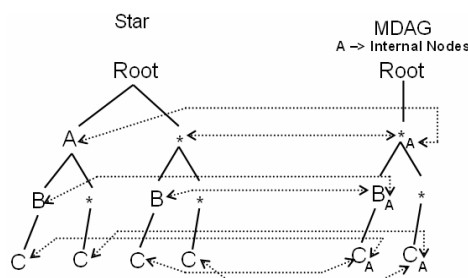
but we can extend this representation if we eliminate one dimension of the main lattice and associate it to all the remaining dimensions directly, forming a DAG. The eliminated dimension produces the internal nodes and the remaining dimensions produce the external nodes. In Figure 1, we illustrate a 3D data cube using the new MDAG representation with its internal nodes and the original star-tree representation. We also map the original star-tree representation to the MDAG representation to facilitate comparisons between the data structures.

A path from the root to an external node, associated or not to an internal node, represents a cube cell. Each external node has 5 fields: (i) an attribute value; (ii) set of aggregate values; (iii) set of internal nodes; (iv) pointer(s) to possible descendant(s) node(s); and (v) pointer(s) to possible sibling(s) node(s). Each internal node has two fields: (i) an attribute value; and (ii) set of aggregate values, similar to the external nodes.

The set of aggregate values permits the computation of simultaneous measures. Each aggregate value is associated to one or more column(s) of the base relation and to a statistic function (MIN, MAX, AVG, etc.).

The extended MDAG representation is lossless, since any cube cell can be found in the extended representation using the following rule: (i) cube cells that aggregate the eliminated dimension (Ex.  $***$ ,  $*B^*$ ,  $**C$ ,  $*BC$  cells) are found in the data structure composed by only external nodes (similar to the star-tree); and (ii) cube cells that set the eliminated dimension value (Ex.  $A**$ ,  $AB^*$ ,  $A*C$ ,  $ABC$  cells) are found in the data structure composed by external and internal nodes. In this paper, we use uppercase letters to represent dimensions (Ex. A, B, C) and lowercase letters to represent tuples or registers (Ex.  $a_1$ ,  $b_1$ ,  $c_1$ ).

In the remaining of this section, we discuss the impact of the wildcard elimination and the external/internal node creation. We also discuss an extended version of the simultaneous aggregation method, proposed in [Xin et al. 2007], and we finalize the section with the detailed MDAG-Cubing algorithm to compute a full or iceberg data cube.



**Figure 1. Cube representations with/without internal nodes**

### 3.1. Wildcard Elimination

Wildcards are used to represent the entire aggregation (\*) or the infrequent attributes. The computation of wildcards is unnecessary if we concatenate a new property, named dimensional ID, to each MDAG node attribute value. This property indicates which dimension the attribute represents. The new attribute value permits each level in the

MDAG data structure to represent any dimension and not just one dimension as the star-tree representation does. Another interesting observation is that the new attribute value does not increase the length of the original attribute value proposed in [Xin et al. 2007]. We can use 32 *bit* or 64 *bit* attribute value.

In Figure 2, we illustrate a 3D data cube using the new attribute value of the MDAG representation (we use the sign '+' to indicate the concatenation between the attribute value and the dimensional ID) and the original attribute value of the star-tree representation. We also map the original star-tree representation to the MDAG representations (with and without internal nodes) to facilitate comparisons between the data structures and to demonstrate that the MDAG cube representations are lossless.

In a star-tree we have, in the worst case,  $1 + \left( \sum_{i=2}^{d-1} \prod_{j=2}^i (C_j + 1) \right)$  wildcards, independent on the fact that they represent an all-value or a star-value. These wildcards produce two negative impacts:

- (i) They increment the number of new nodes in the data structure. The number of new nodes varies from 9 to 17% in low cardinality relations [5-10 distinct values] and from almost 0 to 2% in high cardinality dimensions, independently from the number of dimensions. We consider  $\prod_{i=1}^d (C_i + 1)$  nodes in the MDAG representation and  $\left( \sum_{i=1}^d \prod_{j=1}^i (C_j + 1) \right) - \left( \prod_{i=1}^{d-1} (C_i + 1) \right)$  nodes in the star-tree representation to calculate the wildcard impact;
- (ii) They increase the data structure height in 33%, if we do not consider the internal node impact, or in 50%, otherwise, and this increase is a fundamental problem when we are traversing a data structure multiple times. The height is calculated using the following expression:  

$$\left( \frac{\text{sum\_all\_branch\_sizes}}{\text{num\_branches}} \right).$$

In this section, we argue that the presence of wildcards is not justified, since its elimination requires just the concatenation of the dimensional ID which does not consume memory space and computation effort. Its utilization, on the other hand, increases the number of new nodes and the data structure height.

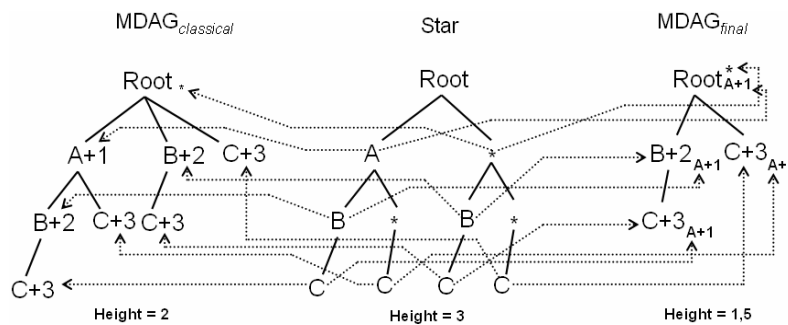
### 3.2. External Nodes and Internal Nodes

The suffix redundancy is a problem in prefixed generic search trees as star-tree. We suggest an efficient solution for the suffix redundancy problem and also for the data structure height and the number of branches, using the notion of external and internal nodes.

We first select the highest cardinality dimension values to form the internal nodes. This dimension is selected, since it produces a cube representation with more internal nodes than any other dimension and this is a fundamental strategy to efficient internal node redundancy elimination. The higher is the number of internal nodes, the higher is the probability of identical aggregate values among them.

In contrast to the internal node decision, we need to select the smallest cardinality dimensions to form the external nodes, since these dimensions produce a denser data structure.

Note that the new MDAG<sub>final</sub> representation, presented in Figure 2, is lossless and it has the same  $\prod_{i=1}^d (C_i + 1)$  nodes as the MDAG<sub>classical</sub> representation (without wildcards only), i.e.,  $\prod_{i=2}^d (C_i + 1)$  external nodes and  $C_1 \prod_{i=2}^d (C_i + 1)$  internal nodes. On the other hand, its height is, on average, 15% smaller and it has no more than half the number of branches of the MDAG<sub>classical</sub> representation, independently from the number of dimensions and the cardinality of each dimension. The reason is based on the fact that the MDAG<sub>classical</sub> representation has, at least,  $C_A$  more branches than the new MDAG representation. If  $C_A = 1$  and  $C_C \leq C_B \leq C_A = 1$ , we have twice more branches than the new MDAG representation. Any other scenario is worse than this one, so we can conclude that the new representation has 50% of the branches or even less, when compared with the MDAG<sub>classical</sub> representation.  $C_A$ ,  $C_B$  and  $C_C$  are the cardinalities of A, B and C, respectively.



**Figure 2. MDAG cube representations with/without wildcards and with/without internal nodes**

In Figure 3, we illustrate the MDAG representation of a tuple  $a_1b_1c_1$ . We have in the extended MDAG representation four internal nodes labeled  $a_1+1$ . If these occurrences have the same aggregate values we need just one internal node  $a_1+1$  to represent the other four, so based on this observation we introduce the internal node redundancy elimination. We use the base relation presented in Figure 4-a to exemplify our ideas.

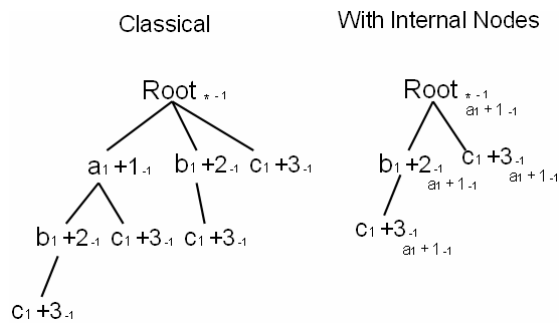
First, we need to select the highest cardinality dimension to form the internal nodes. In our relation of Figure 4-a we have the dimensions B and D as the candidates. To facilitate our explanation we select the dimension D to form our internal nodes.

In the base relation of Figure 4-a we have five tuples. The first two tuples are inserted into the leftmost  $a_1+1$  branch of our representation. The third tuple is inserted into the second  $a_1+1$  branch and the remaining tuples are inserted into the first and the second  $a_2+1$  branches, respectively. The other branches are all aggregations of these five tuples (we explain how to efficiently generate all aggregations in Sections 3.3 and 3.4). Note that, all the external nodes have, at least, one internal node and in order to represent twenty eight external nodes we need only five internal nodes. In this simple relation we have a reduction of 45% when compared with the MDAG<sub>classical</sub>

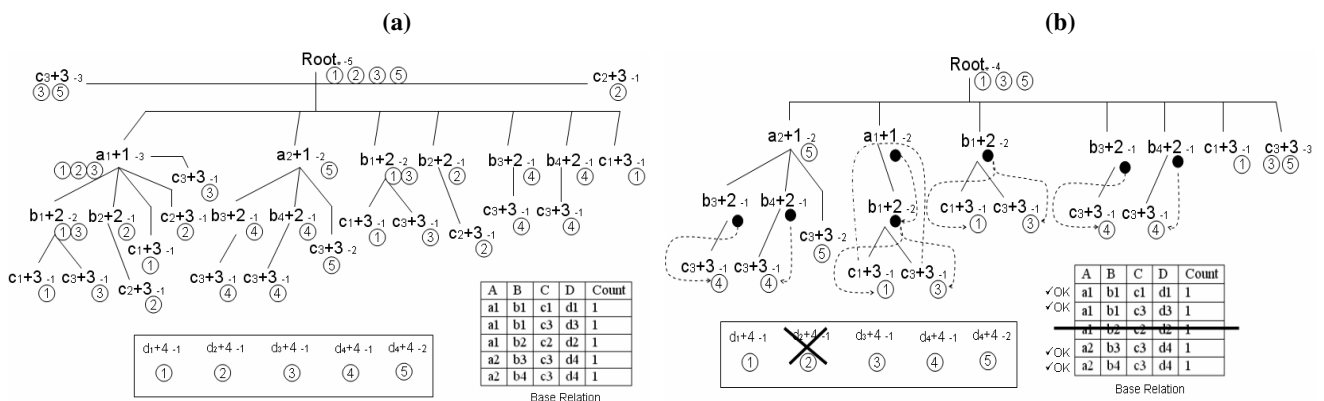
representation and of 60% if we compare with star-tree. This is justified by the fact that we have  $C_1 \prod_{i=2}^d (C_i + 1)$  internal nodes, i.e.,  $C_1$  times the number of external nodes and in this scenario the probability of identical internal nodes becomes higher.

In the worst case the reduced MDAG cube representation has the same  $\prod_{i=1}^d (C_i + 1)$  nodes. Normally, it has  $\epsilon \prod_{i=1}^d (C_i + 1)$  nodes;  $0 < \epsilon < 1$ . In our experiments with synthetic and real datasets  $\epsilon$  varies from 0.1 to 0.3 (70-90% less nodes) when compared with star-tree.

In this section we argued that the adoption of external and internal nodes abstractions in our cube representations is an efficient and simple solution to reduce the cube size and, of course, the cube computation effort. Our MDAG structures are lossless. They represent a fully pre-computed cube without compression, and, hence, it requires neither decompression nor further aggregation when satisfying queries.



**Figure 3. MDAG tuple representations with/without internal nodes**



**Figure 4. (a) A base relation and the respective MDAG cube representation. (b) MDAG cube representation with direct aggregations**

### 3.3. Direct Aggregation Method

In sparse relations about 20% of the group-bys perform very little aggregation (size of nearly 1). These aggregations are simply a projection of the input. If the data were uniform and uncorrelated, then nearly 60% of the group-bys would perform little or no aggregation at all [Beyer et al. 1999].

Based on this observation, we propose an extended version of the simultaneous aggregation method proposed in [Xin et al. 2007]. In Figure 4-b, we illustrate the direct aggregate method using a particular version of the base relation of Figure 4-a. We just eliminate the third tuple to produce all direct aggregations benefits.

In Figure 4-b, the  $a_1+1$  branch has a unique descendant ( $b_1+2$ ) and this descendant has two other descendants ( $c_1+3$  and  $c_3+3$ ). In this situation we do not need to create two external nodes, associate them to  $a_1+1$  and verify their internal nodes, to form the complete  $a_1+1$  aggregations. Instead, we just associate all  $b_1+2$  descendants to  $a_1+1$  node directly. This strategy saves an enormous computational effort and also memory consumption in a sparse space, as our experiments demonstrate.

So far, the direct method is similar to the single path aggregation optimizations proposed in [Xin et al. 2007], but we can extend the idea to be also used with the internal nodes in two scenarios. First, if an external node has just one descendant we can affirm that this external node has the same internal nodes as its descendant, so a simple association can be made. This idea is illustrated in Figure 4-b in the all branches. For example, the node  $b_3+2$  (descendant of the  $a_1+1$ ) has just one descendant ( $c_3+3$ ), so the nodes  $c_3+3$  and  $b_3+2$  have the same internal nodes. Second, if an external node has more than one descendant, but their descendants internal nodes are different, we can apply the same idea when it is just a single descendant. In Figure 4-b this situation is illustrated with the  $b_1+2$  node (descendant of the  $a_1+1$  node). It has two descendants ( $c_1+3$  and  $c_3+3$ ), but their internal nodes are different ( $d_1+4$  and  $d_3+4$ , respectively). In this last situation we just associate the internal nodes of  $c_1+3$  and  $c_3+3$  to  $b_1+2$ . On both scenarios we do not need to verify the internal node pool, what reduces further the computational effort of our approach.

Another interesting observation is the fact that when we are scanning the base relation to produce the base cuboid and the shared dimensions aggregations, associating each external node to its internal nodes, we have more DAG operations than if we use the star-tree representation. Suppose the tuple  $a_1b_1c_1d_1$  being inserted into a star-tree. This tuple demands four tree operations (no matter if it is an insertion or an update), but in the MDAG representations it demands six data structure operations (three to insert  $a_1b_1c_1$  and three to associate  $a_1-d_1$ ,  $b_1-d_1$ ,  $c_1-d_1$ ).

The problem can be efficiently addressed if we just associate the last external node, i.e.  $c_1-d_1$ . This idea enables the MDAG algorithm to consume  $d$  DAG operations, where  $d$  is the number of dimensions. The shared dimension idea continues been used in this context, since the external nodes measures are computed during the first base relation scan.

We need to guarantee that our data structures are lossless, so the remaining associations ( $a_1-d_1$ ,  $b_1-d_1$ ) have to occur. We just make these associations during the simultaneous aggregation, similar to [Xin et al. 2007]. This strategy avoids additional computation effort and we can assure that all ancestral nodes are associated to the correct internal nodes, since an ancestral is just an aggregation of all its descendants.

In this section we argue that in sparse relations we have many nodes with unique descendant node. Due to this fact we develop a direct aggregation method to efficiently associate external nodes and also internal nodes. We verify a problem during the

computation of the base cuboid and the shared dimensions aggregations and we propose a simple solution that guarantees the same DAG operations as star-tree operations, independently if we are using the classical or the reduced MDAG representation.

### **3.4. MDAG Algorithm**

Due to the great amount of construction and destruction of cuboids in the algorithm, memory management becomes an important issue. We use the same strategy proposed in [Xin et al. 2007] to efficiently allocate/de-allocate internal and external nodes.

The complete MDAG algorithm first scans the relation to mount the base *cuboid* and the shared dimensions aggregations. This step is just a sequential scan and a basic MDAG insertion/update, so due to the limited space, we omit the details in this paper. After the base *cuboid* and the shared dimensions aggregations computation, we start the aggregation procedure from the MDAG data structure root.

In Figure 5, we have the detailed MDAG aggregation procedure. First, the algorithm checks if the current external node satisfies the iceberg condition (line 1). If the node does not satisfy it is removed and all its descendants (a set of DAGs) are used to produce new descendants of the current ancestral node (lines 3 and 4). These new descendants are siblings of the eliminated current node, so the MDAG aggregation procedure is also applied to them.

If the current external node satisfies the iceberg condition, the algorithm starts the top-down aggregation method by copying the entire current node descendants to its ancestral descendants (lines 8-12). The MDAG aggregation procedure is also applied to these new current node siblings. An optimization is made when it verifies if the ancestral node has just a single descendant node. If this condition is true, the algorithm just associates all current external node descendants to the ancestral descendants (line 10); otherwise, it clones all current external node descendants (including all its internal nodes) and copy them to the ancestral descendants. This entire copy (line 11) is a costly DAG operation that, in sparse/skewed relations, is frequently avoided. Finally, the depth-first traversal begins (lines 13 and 14) and when it reaches a leaf node, it starts backtracking.

Before traversing back, the algorithm computes the internal nodes of the current external node (lines 16-26), since during the first base relation scan it only relates internal nodes to the leaf external nodes. An optimization is made when it checks if the current external node has just one descendant or if its descendants have different internal nodes (lines 16-18). This optimization eliminates unnecessary internal node pool manipulations and descendant traversals (lines 20-26).

When the algorithm is traversing back, it can also prune infrequent internal nodes (lines 27-29), since the current external node is not be visited again.

## **4. Performance Analysis**

To check the efficiency and the scalability of the proposed algorithm, a comprehensive performance study is conducted by testing MDAG-Cubing algorithms (MDAG has internal nodes redundancy and MDAG-r does not) against the best implementation we could achieve for the Star-Cubing approach (Star is the default algorithm proposed in

[Xin et al. 2003] and Star1 is the algorithm with some optimizations proposed in [Xin et al. 2007]). All the algorithms are coded in Java Tiger 32 bits (JRE 5.0 update 9). We run the algorithms in a Pentium IV 3.73GHz Extreme Edition system with 4GB of RAM. The system runs Linux Fedora Core 5 64 bits (Kernel 2.6.15). All times recorded include both computation and I/O time. Similar to other performance and memory studies in cube computation and representation, all relations can fit in the main memory.

```

BEGIN
  if it is an iceberg cube
    scan R twice, create aux-table with 1D cuboids
    and mount the cuboid C composed by base cells and shared
    dimensions aggregated cells;
  else scan R once and mount the base cuboid C;
  call mdagAggreg(null, MDAG.root);
END

procedure mdagAggreg(ancestralNode, currentNode)
{
  1.   if ((currentNode ≠ root) & (currentNode < min_sup)) {
  2.     //BOTTOM-UP Pruning
  3.     ancestralNode.descendants ← currentNode.descendants; //updates or insertions
  4.     remove currentNode from ancestralNode.descendants;
  5.   } //end if
  6.   else {
  7.     //TOP-DOWN External Nodes Computation
  8.     if (currentNode ≠ root) {
  9.       if (ancestralNode.descendants.size = 1)
 10.        ancestralNode.descendants ← currentNode.descendants; //direct aggr
 11.       else clone all nodes from currentNode.descendants and copy to ancestralNode.descendants;
 12.     } //end if
 13.     for each currentNode.descendants
 14.       mdagAggreg(currentNode, currentNode.oneDescendant);
 15.     //BOTTOM-UP Internal Nodes Computation
 16.     if (currentNode.descendants.size = 1 || currentNode.descendants.internalNodes are ≠)
 17.       for each currentNode.descendants
 18.         currentNode.internalNodes ← currentNode.oneDescendant.internalNodes; //direct aggr
 19.     else
 20.       for each currentNode.descendants {
 21.         for each currentNode.oneDescendant.internalNodes
 22.           if (currentNode.internalNodes has the currentNode.oneDescendant.oneInternalNode) {
 23.             actualize currentNode.oneInternalNode measures;
 24.             actualize internalNodesPool;
 25.           } else currentNode.internalNodes ← currentNode.oneDescendant.oneInternalNode;
 26.         } //end for
 27.       for each currentNode.internalNodes
 28.         if (currentNode.oneInternalNode < min_sup)
 29.           remove oneInternalNode from currentNode.internalNodes;
 30.     } //end else
}

```

**Figure 5. MDAG algorithm**

For the remaining of this section,  $D$  denotes the number of dimensions,  $C$  the cardinality of each dimension,  $T$  the number of tuples in the base *cuboid*,  $M$  the minimum support level, and  $S$  the skew or zipf of the data. When  $S$  is equal to 0, the data is uniform; as  $S$  increases, the data is more skewed.  $S$  is applied to all dimensions.

#### 4.1. Full Cube Computation

The first set of experiments compare Star-Cubing with the MDAG algorithms for full cube computation. The runtime and the memory consumption are compared with respect

to the cardinality (Figures 6 and 7), tuple size (Figures 8 and 9) and dimension (Figures 10 and 11).

In the first experiment, we randomly generate relations with 5 dimensions, varying the cardinality of each dimension from 20 to 100. In the second experiment, we vary the number of tuples from 1 to 5M. Finally, we increase the dimension number from 5 to 8 while keeping the cardinality of each dimension at 10. The tuple size for the cardinality and the dimension experiments are 5M. All the data are uniformly distributed, i.e., skew is 0.

The MDAG and MDAG-r beat the Star-Cubing algorithms in all runtime experiments. The MDAG algorithms run, on average, 15-35% faster than the Star-Cubing. The MDAG memory consumption reduction varies from 75-95% when compared with Star-Cubing. There is a memory reduction in Star1, caused by the shared single paths aggregations, when compared to the Star. Finally, all algorithms suffer from dimension growth (Figure 10), but the low memory consumption of MDAG-r turns it a promise solution to compute high dimensional data cubes.

#### **4.2. Iceberg Cube Computation**

The second set of experiments compares the four algorithms for iceberg cube computation. All the tested algorithms use some form of pruning that exploits the anti-monotonicity of the count measure.

Figures 12 and 13 show the performance with respect to cardinality. The relations have 5M tuples with 5 dimensions and 0 skew. The performance result shows that all algorithms do not suffer from cardinality increase (full or iceberg cube), but the MDAG approach is significantly better. The iceberg MDAG cube computation is more efficient because it reduces the cube representation and adopts the idea of internal and external nodes. When we prune an external node, we also prune all its internal nodes and this is an enormous advantage when compared to the Star-Cubing approach. In Figures 14, 15, 16 and 17, MDAG continues being better than Star-Cubing.

Figures 18 and 19 show what happens when we increase the user threshold (*min\_sup*). The Star-Cubing uses an auxiliary table, named *star-table*, to prune infrequent cells. The *star-table* causes a second relation scan and this is very costly when the relation has a huge number of tuples. Our experiments demonstrate that the *star-table* is useful only when the user threshold or the skew are high. In Figure 18, we set the *min\_sup* to 10000 and only in this situation the *star-table* brings an advantage. The MDAG approach can easily use the auxiliary table idea, but it does not use just it to see how efficient it is.

#### **4.3. Data Skew**

The third experiment tests the four algorithms with respect to data skews. We use zipf to control the skew, varying from 0.5 to 3. The MDAG algorithms have an almost uniform behavior when we vary the data skew (see Figures 20 and 21). The Star-Cubing, on other hand, suffers when the skew is low. The memory consumption difference is high in low skewed datasets and it becomes smaller as the skew increases. The reason is the fact that we have less frequent tuples and high number of single paths in high skewed datasets, so in this scenario the memory consumption becomes similar.

#### **4.4. Real World Dataset**

This dataset is derived from the HYDRO1k Elevation Derivative Database. It is a geographic database developed to provide hydrologic information on a continental scale. In our experiments, we use the dataset of South America. In the hydrologic information fact table, there are five dimensions: slope, compound topographic index, flow directions, latitude and longitude. The cardinalities for the dimensions are 300, 280, 160, 78 and 53, respectively. The fact table includes one measure: Flow Accumulations (FA). The total number of rows in the fact table is 7,845,529. For a full description of the original dataset see <http://edcdaac.usgs.gov/gtopo30/hydro/>.

In Figures 22 and 23 we see a similar behavior to the syntactic datasets, i.e., the MDAG algorithms are 30-50% faster than the Star-Cubing and the MDAG-r consumes about 10% of memory when compared with star-tree.

### **5. Discussion**

In this section, we discuss a few issues related to MDAG-Cubing and point out some research directions.

#### **5.1. Computing Complex Measures**

In this paper, each DAG internal or external node has a set of aggregate values, each of them associated to one or more column(s) of the fact table and to a statistic function. Computing iceberg cube with complex measures, such as `average()`, can be easily included into our algorithms, based on the technique proposed in [Han et al. 2001].

For example, for computing iceberg cube with the condition, “ $\min \text{sup}(c) = k$  and  $\text{average}(c) > v$ ”, for each cell  $c$ , one may store top- $k$  quant-info at MDAG cube representation and use the same technique as the one proposed in [Han et al. 2001], [Lakshmanan, 2002] to perform anti-monotonicity testing to filter out those unpromising nodes during the cube computation process. Computing other complex measures may adopt the similar techniques suggested in [Han et al. 2001].

#### **5.2. Handling Large Databases and the Dimension Increase Problem**

In this section, we separate the problem of low/medium dimension relations, which generates a huge amount of cells that cannot fit in main memory, from high dimension relations that cannot be efficiently computed by the current MDAG and Star-Cubing approaches. For the first problem, we can achieve an initial solution if we use a dimension to segment the MDAG cube representation. Instead of one data structure, we have a set of substructures labeled by the eliminated dimension values. This solution is implemented in a parallel version of MDAG approach, but due to the limited space, we omit the details in this paper. One may also consider the case that even the specific substructure may not fit in memory. For this situation, the projection-based preprocessing proposed in [Han et al. 2000] can be an interesting solution.

We have many approaches that address solutions to the dimension increase problem, but none of them can solve the fundamental problem of number of cells and runtime. In [Li et al. 2004] the authors proposed one of the more efficient approaches for high dimension datasets with medium number of tuples (around  $10^6$  tuples). We

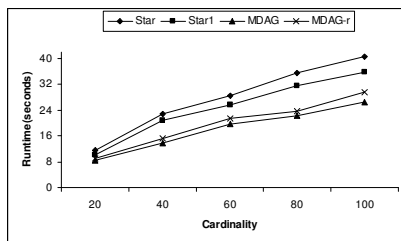
believe that the adoption of this idea in conjunction with our idea can produce an efficient MDAG approach for high dimensional data cube computation, but the row-based approach needs to be reformulated to guarantee the computation of cube with high number of tuples.

## 6. Conclusions

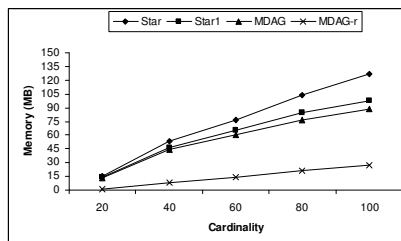
For efficient cube computation in various data distributions, we have proposed a new approach named MDAG-Cubing. This approach addresses an efficient solution for the costly multiple data structure traversals Star-Cubing problem. The solution is based on three main properties: (i) it eliminates both wildcards (\* and star) from its cube representation; (ii) it significantly reduces the suffix redundancy, the DAG height and the number of DAG branches, using the notion of internal and external nodes; and (iii) it directly aggregates external nodes and also internal nodes when the number of descendants is unique or the internal nodes descendants are different. These characteristics enable MDAG-Cubing to reduce the memory consumption and runtime when computing skewed or sparse relations.

Our performance studies demonstrate that MDAG-Cubing is a promising algorithm. In general, the MDAG algorithms run 25-50% faster than the Star-Cubing algorithm. The memory consumption decreases drastically (70-90% less memory).

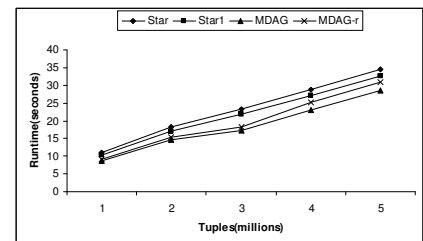
There are many interesting research issues to further extend MDAG-Cubing. We believe that efficient methods to reduce the external node redundancy, to update the MDAG data structure and to enable MDAG-Cubing to compute high dimension relations are required. Parallel cube computation and discover-driven methods can be made using the MDAG data structure.



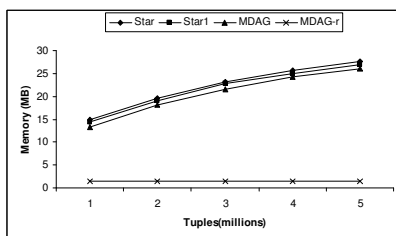
**Figure 6. Full Cube: D=5, T=5M, S=0, M=1**



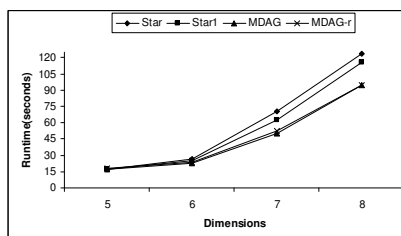
**Figure 7. Full Cube: D=5, T=5M, S=0, M=1**



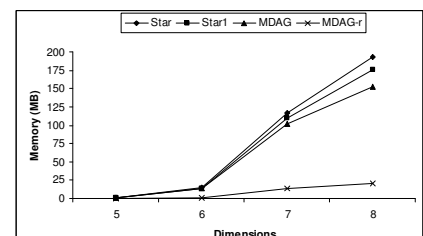
**Figure 8. Full Cube: D=5, C=20, S=0, M=1**



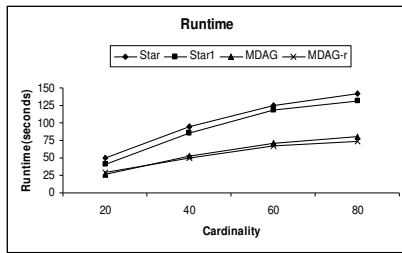
**Figure 9. Full Cube: D=5, C=20, S=0, M=1**



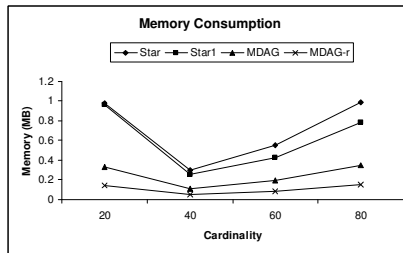
**Figure 10. Full Cube: T=5M, C=10, S=0, M=1**



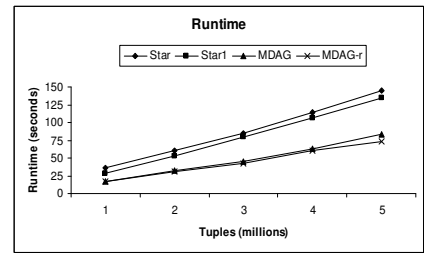
**Figure 11. Full Cube: T=5M, C=10, S=0, M=1**



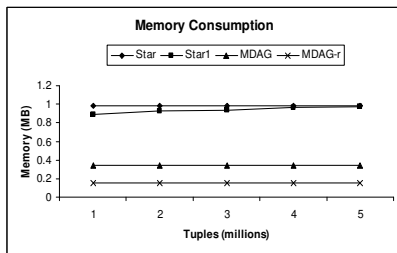
**Figure 12. Iceberg Cube: D=5, T=5M, S=0, M=100**



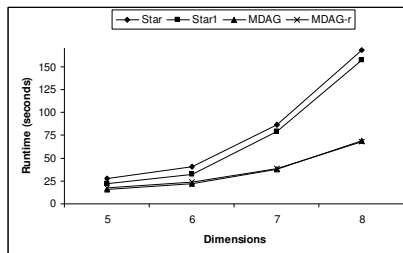
**Figure 13. Iceberg Cube: D=5, T=5M, S=0, M=100**



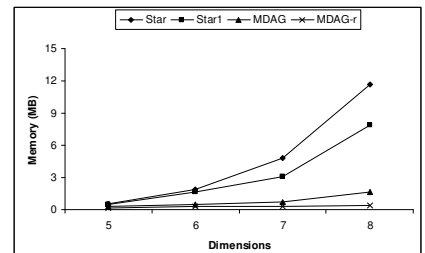
**Figure 14. Iceberg Cube: D=5, C=80, S=0, M=100**



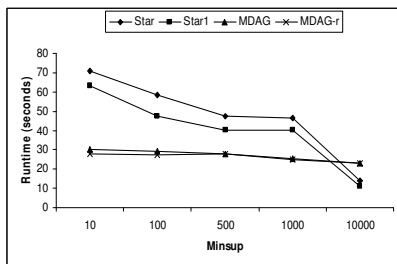
**Figure 15. Iceberg Cube: D=5, C=80, S=0, M=100**



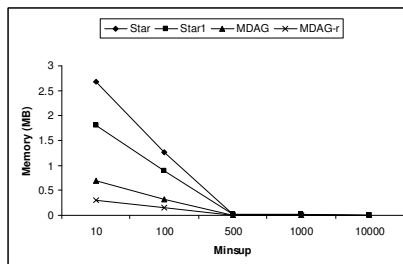
**Figure 16. Iceberg Cube: T=5M, C=10, S=0, M=100**



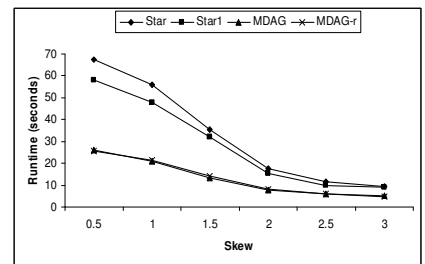
**Figure 17. Iceberg Cube: T=5M, C=10, S=0, M=100**



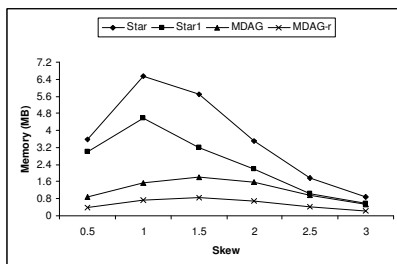
**Figure 18. Iceberg Cube: D=6, T=1M, C=100, S=0**



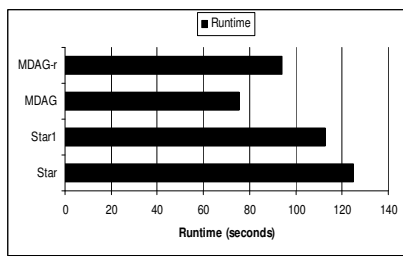
**Figure 19. Iceberg Cube: D=6, T=1M, C=100, S=0**



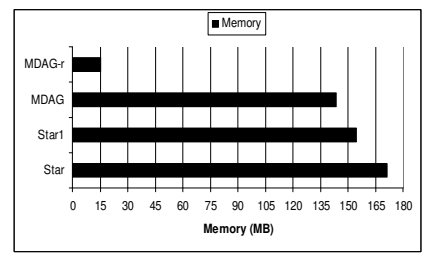
**Figure 20. Iceberg Cube: D=6, T=1M, C=100, M=10**



**Figure 21. Iceberg Cube: D=6, T=1M, C=100, M=10**



**Figure 22. Real Dataset Runtime**



**Figure 23. Real Dataset Memory**

## References

Baralis, E., Paraboschi, S. and Teniente, E. "Materialized view selection in a multidimensional database", VLDB '97, pages 156–165, East Sussex- San Francisco, Aug. 1998. Morgan Kaufmann.

**SBBD 2007**  
*XXII Simpósio Brasileiro de Banco de Dados*

- Barbará, D. and Sullivan, M. “Quasi-cubes: Exploiting approximations in multidimensional databases”, *SIGMOD Record*, 26(3):12–17, 1997.
- Beyer, K. and Ramakrishnan, R. “Bottom-up computation of sparse and Iceberg CUBEs”, *SIGMOD*, 28(2):359–371, 1999.
- Chen, Y., Dong, G., Han, J., Wah, B.W. and Wang, J. “Multi-Dimensional Regression Analysis of Time-Series Data Streams”, *VLDB'02*.
- Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F. and Pirahesh, H. “Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub totals”, *Data Min. Knowl. Discov*, 1(1):29–53, 1997.
- Han, J., Pei, J., Dong, G. and Wang, K. “Efficient computation of iceberg cubes with complex measures”, volume 30, 2 of *SIGMOD Record*, pages 1–12, New York, May 21–24 2001. ACM Press.
- Han, J., Pei, J. and Yin, Y. “Mining frequent patterns without candidate generation”, In *Proceedings of the 2000 ACM SIGMOD: May 16–18, 2000, Dallas, Texas*, volume 29(2) of *SIGMOD Record*, pages 1–12, pub-ACM:adr, 2000.
- Lakshmanan, L.V.S., Pei, J. and Han, J. “Quotient cube: How to summarize the semantics of a data cube”, In *VLDB*, pages 778–789. Morgan Kaufmann, 2002.
- Li, X., Han, J. and Gonzalez, H. “High-dimensional OLAP: A minimal cubing approach”, In M. A. Nascimento, *VLDB'04*, pages 528–539. Morgan Kaufmann, 2004.
- Shukla, A., Deshpande, P. and Naughton, J. F. “Materialized view selection for multidimensional datasets”, In *VLDB '98*, pages 488–499, East Sussex – San Francisco, Aug. 1998. Morgan Kaufmann.
- Sismanis, Y., Deligiannakis, A., Roussopoulos, N. and Kotidis, Y. “Dwarf: shrinking the petacube”, *SIGMOD Conference*, pages 464–475. ACM, 2002.
- Vitter, Wang, and Iyer. “Data cube approximation and histograms via wavelets”, In *ACM CIKM. ACM, SIGIR, and SIGMIS*, 1998.
- Wang, W., Lu, H., Feng, J. and Yu, J. X. “Condensed cube: An efficient approach to reducing data cube size”, In *ICDE*, pages 155–165. IEEE Computer Society, 2002.
- Xin, D., Han, J., Li, X. and Wah, B.W. “Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration”, *Proc. 2003 Int. Conf. on Very Large Data Bases (VLDB'03)*, Berlin, Germany, Sept. 2003.
- Xin, D., Han, J., Li, X., Shao, Z. and Wah, B.W. “Computing Iceberg Cubes by Top-Down and Bottom-Up Integration: The StarCubing Approach”, *IEEE Transactions on Knowledge and Data Engineering*, 19(1): 111-126, 2007.
- Xin, D., Shao, Z., Han, J., and Liu, H. “C-cubing: Efficient computation of closed cubes by aggregation-based checking”, In *ICDE'06*, page 4. IEEE Computer Society, 2006.
- Zhao, Y., Deshpande, P., and Naughton, J. F. “An array-based algorithm for simultaneous multidimensional aggregates”, In *ACM SIGMOD*, pages 159–170, Tucson, Arizona, 13–15 June 1997.