

# The Dwarf Data Cube Eliminates the High Dimensionality Curse

**Yannis Sismanis**

University of Maryland  
Institute for Advanced  
Computer Studies  
isis@cs.umd.edu

**Nick Roussopoulos**

University of Maryland  
Institute for Advanced  
Computer Studies  
nick@cs.umd.edu

## Abstract

The data cube operator encapsulates all possible groupings of a data set and has proved to be an invaluable tool in analyzing vast amounts of data. However its apparent exponential complexity has significantly limited its applicability to low dimensional datasets. Recently the idea of the *dwarf data cube model* was introduced, and showed that high-dimensional “dwarf data cubes” are orders of magnitudes smaller in size than the original data cubes even when they calculate and store every possible aggregation with 100% precision.

In this paper we present a surprising analytical result proving that the size of dwarf cubes grows *polynomially* with the dimensionality of the data set and, therefore, a full data cube at 100% precision is not inherently cursed by high dimensionality. This striking result of polynomial complexity reformulates the context of cube management and redefines most of the problems associated with data-warehousing and On-Line Analytical Processing. We also develop an efficient algorithm for estimating the size of dwarf data cubes before actually computing them. Finally, we complement our analytical approach with an experimental evaluation using real and synthetic data sets, and demonstrate our results.

## 1 Introduction

The data cube operator is an analytical tool which provides the formulation for aggregate queries over categories, rollup/drilldown operations and cross-tabulation. Conceptually the data cube operator encapsulates all possible multi-dimensional groupings and its an invaluable tool to applications that need analysis on huge amounts of data like decision support systems, business intelligence and data mining.

Such applications need very fast query response on mostly ad-hoc queries that try to discover trends or patterns in the data set.

However the complexity of the data cube increases *exponentially* with the number of dimensions and most approaches are unable to compute and store but small low-dimensional data cubes. After the introduction of the data cube in [6] an abundance of research followed for dealing with the exponential complexity of the data cube. The main ideas can be classified as either a cube sub-setting (partial materialization) [7, 8, 17] or storing the full cube but with less precision (approximation or lossy models) [1, 18]. However, all these techniques do not directly address the problem of space complexity. Furthermore, all problems associated with the data cube itself appeared to be quite difficult, from computing it [2, 4, 14, 20, 3, 12], storing it [9, 5], querying and updating it[13]. Even problems that appear simpler, like obtaining estimates on the cube size, is actually quite hard and needs exponential memory with respect to the dimensionality [15] in order to obtain accurate results.

Currently the most promising approaches for handling large high-dimensional cubes lie in the context of *coalesced* data cubes[16, 11, 19]. In [16] we demonstrate that the size of the dwarf data cube even when they compute, store and index every possible aggregate for all group-bys is orders of magnitudes smaller than what expected. The coalescing discovery published for the first time in the dwarf data cube model [16], completely changed the perception of a data cube from a unordered collection of distinct groupings into a complex network of interleaved groupings and aggregates that eliminate both *prefix* and *suffix redundancies*. It is these redundancies and their elimination that fuse the exponential growth of the size of high dimensional full cubes and dramatically condense their store without loss in precision.

To help clarify the basic concepts, let us consider a cube

with three dimensions. In Table 1 we present such a toy dataset for the dimensions Store, Customer, and Product with one measure Price.

| Store | Customer | Product | Price |
|-------|----------|---------|-------|
| S1    | C2       | P2      | \$70  |
| S1    | C3       | P1      | \$40  |
| S2    | C1       | P1      | \$90  |
| S2    | C1       | P2      | \$50  |

Table 1: Fact Table for Cube Sales

The size of the cube is defined as the number of the tuples it contains, which essentially corresponds to the sum of the tuples of all its  $2^3$  views. The size of the dwarf is defined similarly as the total number of tuples it contains, *after* data coalescing. For example, for the fact table in Table 1 and the aggregate function *sum* we have a cube size of 23 tuples, while the dwarf size is just 9 tuples as depicted in Table 2. The redundancy of the cube is eliminated in the dwarf cube and the coalesced areas are only stored once. For example, the aggregate \$70 appears in total of five tuples, (S1|ALL,C2,P2|ALL) and (S1,ALL,P2), in the cube and it is coalesced in just one tuple in the dwarf.

| no | Coalesced                      | Price |
|----|--------------------------------|-------|
| 1  | (S1 ALL,C2,P2 ALL) (S1,ALL,P2) | \$70  |
| 2  | (S1 ALL,C3,P1 ALL) (S1,ALL,P1) | \$40  |
| 3  | (S1,ALL,ALL)                   | \$110 |
| 4  | (S2 ALL,C1,P1) (S2,ALL,P1)     | \$90  |
| 5  | (S2 ALL,C1,P2) (S2,ALL,P2)     | \$50  |
| 6  | (S2 ALL,C1,ALL)                | \$140 |
| 7  | (ALL,ALL,P1)                   | \$130 |
| 8  | (ALL,ALL,P2)                   | \$120 |
| 9  | (ALL,ALL,ALL)                  | \$250 |

Table 2: Dwarf Cube Tuples

In this paper we address the problem of the size of the dwarf data cube and show that it's complexity is:

$$O\left(d^{\log_C T} \frac{T}{(\log_C T)!}\right) = O\left(T^{1+1/\log_d C}\right)$$

where  $d$  is the number of dimensions,  $C$  is the cardinality of the dimensions and  $T$  is the number of tuples. This surprising result shows that, unlike the case of non-coalesced cubes which grow exponentially fast with the dimensionality, the 100% accurate and complete (in the sense that it contains all possible aggregates) dwarf representation only

grows *polynomially* fast. In other words, if we keep the number of tuples in the fact table constant and start increasing the dimensionality of the fact table (by horizontally expanding each tuple with new attributes) then the size of the dwarf increases only polynomially. The first form shows that the dimensionality  $d$  is raised to  $\log_C T$  which does not depend on  $d$  and is actually quite small for most realistic datasets<sup>1</sup>.

The second form of the complexity shows that the Dwarf size is polynomial w.r.t to the number of tuples of the data set  $T$ , which is raised to  $1 + 1/\log_d C$  (and is very close to 1 for most realistic datasets<sup>2</sup>). In other words, if we keep the dimensionality of the fact table constant and start appending new tuples, then the size of the dwarf increases polynomially (and almost linearly). These results change the current state of the art in data-warehousing because it allows to scale up and be applicable to a much wider area of applications.

In addition we extend our analysis to cubes with varying cardinalities per dimension and we provide a linear -w.r.t to the dimensionality- algorithm which can be used to estimate the size of a dwarf cube based only on these cardinalities without actually computing the dwarf cube. Such estimates are invaluable for data-warehouse/OLAP administrators who need to allocate the storage for the dwarf before initiating its computation. Current approaches [15] cannot be applied to high-dimensional data cubes, not only because they require an exponential amount of work per tuple and exponential amount of memory but mostly because they cannot be extended to handle coalesced cubes.

In particular in this paper we make the following contributions:

1. We formalize and categorize the redundancies found in the structure of the data cube into sparsity and implication redundancies
2. For the sparsity redundancies, we provide an analysis on the size of the dwarf cube, and show that it scales only polynomially w.r.t to the number of dimensions.
3. We complement our analytical contributions with an algorithm and an experimental evaluation using both synthetic and real data sets and we show that in real data sets, the size of the dwarf cube is even smaller due to implication redundancies.

Our work provides the *first* analytical results showing that a full and 100% precision data cube is not inherently ex-

<sup>1</sup>For example for a data set of 25 million tuples and a cardinality of 5,000,  $\log_C T = 2$

<sup>2</sup>I.e., for a dimensionality of 30 and a cardinality of 5,000,  $1 + 1/\log_d C \approx 1.4$

ponential in size and that an effective coalescing data cube model can reduce its size to realistic values. Therefore, we believe it has not only theoretical but also practical value for OLAP.

The remainder of the paper is organized as follows: In Section 2 we differentiate between prefix and suffix redundancies and show that suffix redundancies are by far the most dominant factor that affects coalesced cubes. Section 3 categorizes suffix redundancies based on the sparsity of the fact table or the implications between values of the dimensions. In Section 4 we introduce the notion of basic partitioned node and we use it to analyze the coalesced cube structure. In Section 5 we present an algorithm that can be used to estimate the size of a coalesced cube given only the cardinalities of each dimension and in Section 7 we show an evaluation on both synthetic and real data sets. Finally the conclusions are summarized in Section 8.

## 2 Redundancies

In this section we formalize the redundancies found in the structure of the cube and explain their extend and significance.

### 2.1 Prefix Redundancy

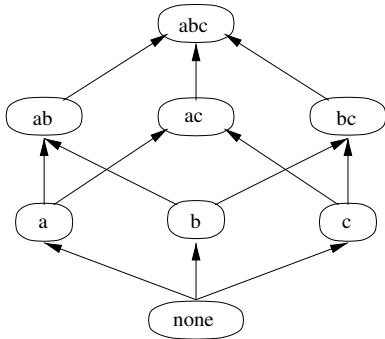


Figure 1: Lattice for the ordering  $a, b, c$

This redundancy is the first that has been identified and can be used to build indexes over the structure of the cube. The idea is easily visualized in the lattice representation of the cube. For example, in Figure 1, one can observe that half the group-by’s share the prefix  $a$ . We can exploit this by just storing the corresponding values just once and avoid replicating the same values over all views(prefix-reduction). By generalizing this to other prefixes (like for example to prefix  $b$ , which appears to one fourth of the views) we can

reduce the amount of storage required to store the tuples of the cube.

**Lemma 1** *The total number of tuples of the cube is not affected by prefix redundancy, only the storage required to store each tuple is reduced.*

This lemma essentially says that the prefix-reduced cube still suffers from the dimensionality curse, since we have to deal with every single tuple of the cube. The benefits of the prefix-reduction are therefore quickly rendered impractical even for medium dimensional cubes.

### 2.2 Suffix Redundancy

In this section we formally define the suffix redundancy and we give examples of different suffix redundancies.

**DEFINITION 1** *Suffix Redundancy occurs when a set of tuples of the fact table contributes the exact same aggregates to different groupings. The operation that eliminates suffix redundancies is called **coalescion**. The resulting cube is called **coalesced cube**.*

**EXAMPLE 1** *Suffix redundancy can occur for just a single tuple: In the fact table of Table 1, we observe that the tuple:*

$$\langle S1 C2 P2 \$70 \rangle$$

*contributes the same aggregate \$70 to two group-bys: (Store, Customer) and (Customer). The corresponding tuples are:*

|                              |                           |
|------------------------------|---------------------------|
| (Store, Customer)            | (Customer)                |
| $\langle S1 C1 \$70 \rangle$ | $\langle C2 \$70 \rangle$ |

**EXAMPLE 2** *We must point out that suffix redundancy does not work only on a per-tuple basis, but most importantly it extends to whole sub-cubes, for example the sub-cube that corresponds to the tuples:*

$$\langle S2 C1 P1 \$90 \rangle, \langle S2 C1 P2 \$50 \rangle$$

*contributes the same aggregates to sub-cubes of (Store, Product), (Customer, Product), (Store), (Customer) :*

The reason that whole sub-cubes can be coalesced is the *implication* between values of the dimensions. In our example,  $C1$  implies  $S2$ , in the sense that customer  $C1$  only

|                              |                              |
|------------------------------|------------------------------|
| (Store,Product)              | (Customer,Product)           |
| $\langle S2 P1 \$90 \rangle$ | $\langle C1 P1 \$90 \rangle$ |
| $\langle S2 P2 \$50 \rangle$ | $\langle C1 P2 \$50 \rangle$ |

|                            |                            |
|----------------------------|----------------------------|
| (Store)                    | (Customer)                 |
| $\langle S2 \$140 \rangle$ | $\langle C1 \$140 \rangle$ |

buys products from store  $S2$ . Dwarf is the only technique that manages to identify such whole sub-cubes as redundant and coalesce the redundancy from *both* storage and computation time, *without* calculating any redundant sub-cubes. For comparison, the condensed cube[19] can only identify redundant areas only tuple-by-tuple, and QC-Trees[11] have to compute first all possible sub-cubes and then check if coalescing can occur.

Such suffix redundancies demonstrate that there is significant overlap over the aggregates of different groupings. The number of tuples of the coalesced cube, where coalesced areas are only store once is much smaller than the size of the cube, which replicates such areas over different groupings.

**DEFINITION 2** *The size of a cube is the sum of the tuples of all its views. The size of a coalesced cube is the total number of tuples after the coalescion operation.*

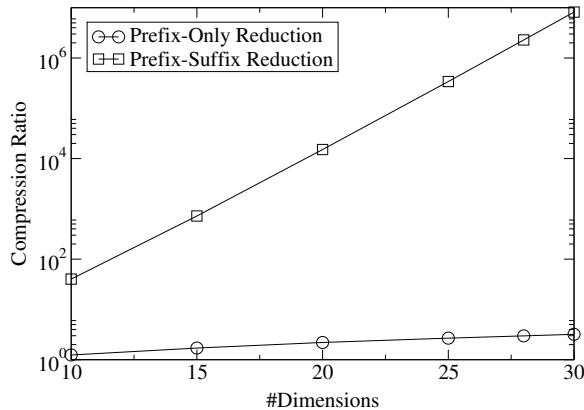


Figure 2: Compression vs. Dimensionality

Prefix redundancy works in harmony with suffix redundancy by eliminating common prefixes of coalesced areas. A comparison between these redundancies is demonstrated in Figure 2, where we depict the compression ratio achieved by storing all the tuples of a cube exploiting in the first case just the prefix redundancies and in the second both prefix and suffix redundancies w.r.t to the dimensionality of the

dataset. We used a dataset with a varying number of dimensions, a cardinality of 10,000 for each dimension and a uniform fact table of 200,000 tuples. It is obvious that in high-dimensional datasets the amount of suffix redundancies is many orders of magnitudes more important the prefix redundancies.

### 3 Coalescing Categories

In this section we categorize suffix redundancies in *sparsity* and *implication* redundancies. We use the Dwarf model[16] -a summary is in the appendix- in order to define and visualize the redundancies. In the rest of the paper we will use the Dwarf model for analysis, but our approach can be applied to other coalesced cube approaches[11, 19].

#### 3.1 Sparsity Coalescing

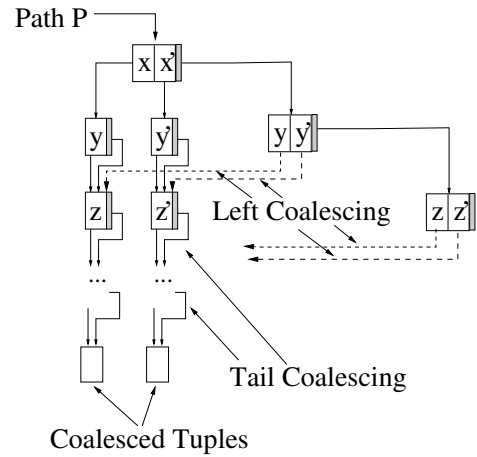


Figure 3: Sparsity Coalescings

In Figure 3 we depict two types of suffix redundancies due to the sparsity of the dataset. Lets assume that a path  $\langle P \rangle$  leads to a sparse area and that for the paths  $\langle P x \rangle$  and  $\langle P x' \rangle$  there is only one tuple due to the sparsity of the cube. We differentiate to two different types of coalescing based on the nature of the path  $P$ .

**DEFINITION 3** *Tail coalescing happens on all groupings that have  $\langle P x \rangle$  as a prefix, where path  $\langle P x \rangle$  leads to a sub-dwarf with only one fact tuple and path  $P$  does not follow any ALL pointers.*

**EXAMPLE 3** *In Figure 3, since there is only one tuple in the area  $\langle P x \dots \rangle$  then all the group-bys that have  $\langle P x \rangle$  as*

a prefix (i.e.  $\langle P x ALL z \dots \rangle$ ,  $\langle P x y ALL \dots \rangle$  etc.) share the same aggregate.

**DEFINITION 4** *Left coalescing* occurs on all groupings with prefix  $\langle P ALL y \rangle$ , where path  $\langle P ALL y \rangle$  leads to a sub-dwarf with only one tuple. In this case,  $P$  may follow *ALL* pointers.

**EXAMPLE 4** *Left coalescing complements tail coalescing* and in Figure 3 we depict the case where  $\langle P ALL y \dots \rangle$  is redundant and corresponds to  $\langle P x y \dots \rangle$ . The same is observed for  $\langle P ALL ALL z \rangle$  and  $\langle P ALL ALL z' \rangle$ .

Areas with just one tuple (like  $\langle P xy \rangle$  and  $\langle P x'y' \rangle$ ) therefore produce a large number of redundancies in the structure of the cube. The difference between tail and left coalescing is two-fold:

- Paths that lead to tail coalescing *do not follow* any *ALL* pointers while in left coalescing the paths follow at least one *ALL* pointer -the one immediately above the point where coalescing happens-.
- Tail coalescing introduces one coalesced tuple in the dwarf, while left introduces no coalesced tuples.

In our analysis we only consider these two types of coalescing (tail and left) and we show that their effect is so overwhelming that the exponential nature of the cube reduces into a polynomial dwarf.

### 3.2 Implication Coalescing

The final type of coalescing that the dwarf very efficiently performs is called *implication-coalescing*. The sparsity-coalescing types defined in Section 3.1 work only in sparse areas of the cube where a single tuple exists. The implication-coalescing complements these redundancies by coalescing *whole sub-cubes*. For example, for the fact table in Table 1 we observe that  $C1$  implies  $S2$  -in the sense that customer  $C1$  only buys products from  $S2$ . This fact means that *every* grouping that involves  $C1$  and  $S2$  is essentially exactly the same with the groupings that involve  $C1$ . This redundancy can be depicted in Figure 4

The implication coalescing is the generalization of left-coalescing when implications between the values of dimension occur. Such implications are very apparent in real datasets and -since we do not consider those in our analysis- they are the reason that in the experiments section we *over-estimate* the size of the coalesced cube for real data sets.

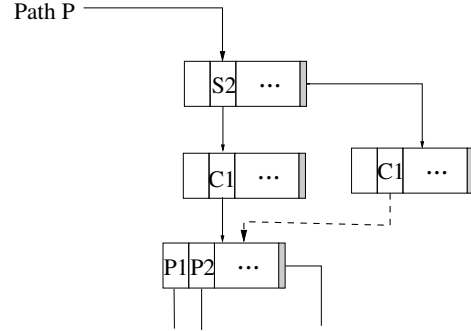


Figure 4: Implication Coalescing, where  $C1 \rightarrow S2$

## 4 Basic Partitioned Node

In this section we formulate coalesced cube structure using the dwarf model by first introducing the *basic partitioned node* and then by building the rest of the coalesced cube around it -by taking into account both tail and left coalescing-.

Assume a uniform fact table with  $d$  dimensions, where each dimension has a cardinality of  $C = l!$  and that there are  $T = C$  tuples. The root node of the corresponding dwarf is depicted in Figure 5, where the node has been partitioned<sup>3</sup> into  $l$  groups. We refer to such a node as the *basic partitioned node*. Group  $G_0$  contains cells that get no tuples at all, group  $G_1$  contains cells that get exactly one tuple, group  $G_2$  contains cells that each one gets exactly two tuples, etc.

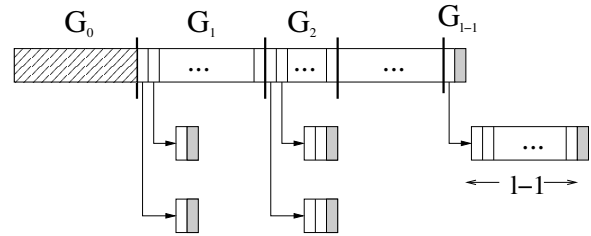


Figure 5: Node partitioned in groups where each cell in group  $G_z$  gets exactly  $z$  tuples

**Lemma 2** *From a collection of  $C$  items, if we uniformly pick an item and repeat  $T$  times, then the probability that we pick one item exactly  $z$  times is:*

$$P_z(C, T) = \frac{\binom{T}{z}}{(C-1)^z} e^{-T/C}$$

<sup>3</sup>for this analysis we relax the property of the dwarf, where the cells inside a node are lexicographically sorted

[Proof: The probability that we will pick one item exactly  $z$  times is:

$$P_z(C, T) = \binom{T}{z} 1/C^z (1 - 1/C)^{T-z} =$$

$$= \binom{T}{z} 1/C^z (C - 1)^{-z} / C^{-z} (1 - 1/C)^T$$

where the quantity  $(1 - 1/C)^T$  can be approximated by  $e^{-T/C}$  and the binomial  $\binom{T}{z}$  corresponds to the number of different ways the product  $1/C^z (1 - 1/C)^{T-z}$  can be written. ]

By applying lemma 2 to the partitioned dwarf node we get by substituting  $T = C$ :

**Lemma 3** A group  $G_z$  of a basic partitioned node, where  $z = 0 \dots l - 1$ , contains  $\approx \frac{C}{z!} e^{-1}$  cells that get exactly  $z$  tuples each

In Figure 5 we depict that the dominated nodes of a group  $G_z$  have exactly  $z$  cells. From lemma 3 we know that exactly  $z$  tuples are associated with each cell of group  $G_z$  and from the independence assumption we have that the probability that a key is duplicated for these tuples is  $1/C^2$  with an expected number of duplicated keys  $z/C^2$ . Even for  $z = l$ , we expect  $l/(l!)^2 \ll 1$  duplicate cells.

### 4.1 Left Coalesced Areas

In this section we deal with areas of the dwarf that are reachable through paths that follow ALL pointers. These areas have the possibility of left coalescing and as we'll show they are dominated by such redundancies.

In Figure 6 we show a basic partitioned node which corresponds to a path  $P$  that follows at least one ALL pointer and that it corresponds to a subset of the fact table with  $T = C$  tuples. We refer to the corresponding sub-dwarf as *left-coalesced sub-dwarf* and we show that it introduces a "small" number of new coalesced tuples. Obviously cells in group  $G_0$  that get no tuples offer no tuples at all. Cells in group  $G_1$  that get only a single tuple, left-coalesce to other tuples in the structure and offer no aggregation. This is the reason we differentiate between paths that follow at least one ALL pointer and those which do not. Cells in groups  $G_2, G_3, \dots, G_{l-1}$  introduce only a single aggregate per cell.

To help clarify this, consider a cell in group  $G_2$ . Since there are two fact tuples associated with this cell (by definition) there are two paths  $\langle P x \rangle$  and  $\langle P x' \rangle$  that correspond to these two tuples. Since the path  $P$  follows at least one

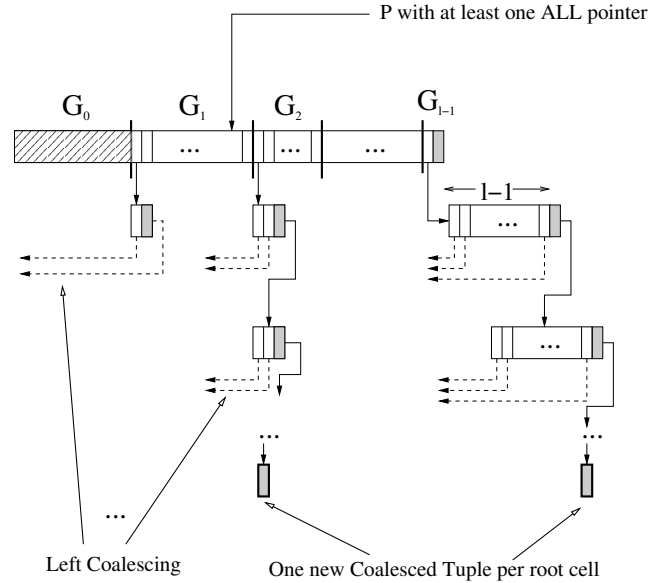


Figure 6: Left-Coalesced partitioned node with  $T = C$

ALL pointer, the *exact same tuples* appear with another path  $Q$  that does not follow any ALL pointer, and therefore paths  $\langle P x \rangle$  and  $\langle P x' \rangle$  coalesce to  $\langle Q x \rangle$  and  $\langle Q x' \rangle$ . The only aggregate that this sub-dwarf introduces is the aggregate of these two tuples (located at the leaf nodes). The same holds for all groups  $G_2, G_3, \dots, G_{l-1}$  and therefore the number of new coalesced tuples that a left-coalesced sub-dwarf with  $d$  dimensions and  $T = C$  fact tuples introduces is (by using lemma 3):

$$NLeft(T = C, d, C) = a_0 \cdot C \cdot d + 1$$

where  $a_0 = (e - 2)/e$

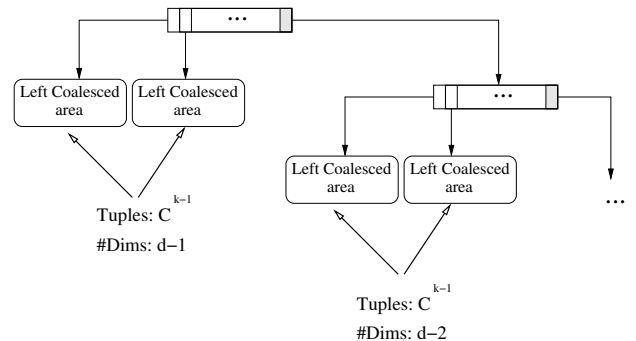


Figure 7: Left-Coalesced partitioned node with  $T = C^k$

We can extend our analysis to the general case where  $T = C^k$ ,  $k = \log_C T$  in the way that is depicted in Figure 7. By induction one can easily prove that:

**Lemma 4** *The number of new coalesced tuples that a left-coalesced area introduces is:*

$$\begin{aligned}
 NLeft(T = C^k, d, C) &= \\
 &= C \cdot \sum_{i=1}^{d-1} NLeft(T = C^{k-1}, d-i, C) + 1 = \\
 &= a_0 C^k \binom{d}{k} + \sum_{i=1}^{k-1} C^{k-i} \binom{d}{k-i} + 1
 \end{aligned}$$

## 4.2 Tail Coalesced Areas

In this section we deal with areas that are reachable through paths that do not follow any ALL pointers. These areas have less chances for left-coalescing but as will show the amount of coalescing is still very significant.

In Figure 8 we show a basic partitioned node which corresponds to a path  $P$  that *does not* follow any ALL pointers and that it corresponds to a subset of the fact table with  $T = C$  tuples. We refer to the corresponding sub-dwarf as *tail-coalesced sub-dwarf* and we count the number of coalesced tuples it introduces. As in the left-coalesced case, cells in group  $G_0$  that get no tuples offer no tuples at all. Cells in group  $G_1$  that get only a single tuple, offer just a single aggregate, due to tail coalescing. Cells in groups  $G_z$ , where  $z = 2, \dots, l-1$  introduce  $z+1$  coalesced tuples, the  $z$  tuples of the fact table plus their aggregation. The number of coalesced tuples a tail-coalesced sub-dwarf with  $d$  dimensions and  $T = C$  fact tables introduces is:

$$NTail(T = C, d, C) = b_0 C + a_0 C(d-1) + 1$$

where  $a_0 = (e-2)/e$  and  $b_0 = (2e-2)/e$ .

We can extend our analysis to the general case where  $T = C^k$ ,  $k = \log_C T$  in the way that is depicted in Figure 9. Using induction we prove that:

**Lemma 5** *The number of new coalesced tuples that a left-coalesced area introduces is:*

$$\begin{aligned}
 NTail(T = C^k, d, C) &= \\
 &= C \cdot NTail(C^{k-1}, d-1, C) + \sum_{i=2}^{d-1} NLeft(C^{k-1}, d-i, C) = \\
 &= a_0 C^k \left[ \binom{d}{k} - 1 \right] + \sum_{i=1}^k C^{k-i} \left[ \binom{d}{k-i} - 1 \right] + b_0 C^k
 \end{aligned}$$

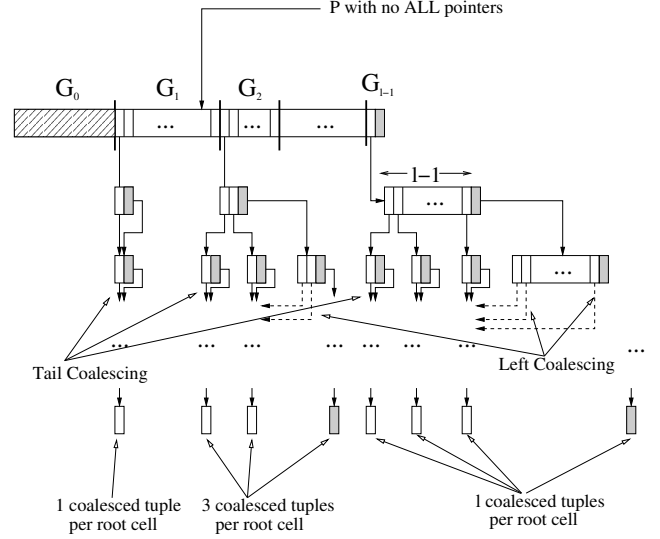


Figure 8: Tail-Coalesced partitioned node with  $T = C$

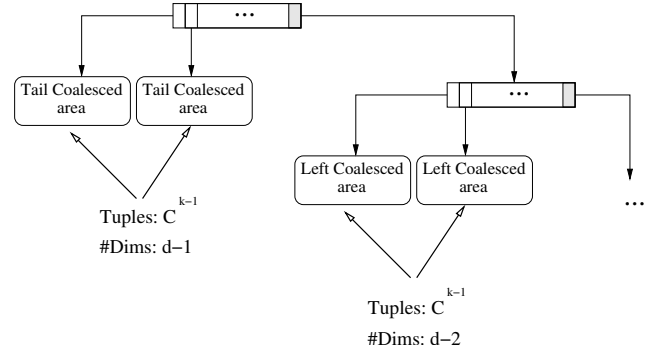


Figure 9: Tail-Coalesced partitioned node with  $T = C^k$

## 4.3 Total Dwarf Size

The analysis for the tail coalesced areas gives the total number of coalesced tuples for the full dwarf with  $d$  dimensions, cardinality  $C$  per dimension and  $T$  fact table tuples<sup>4</sup>. Lemma 5 gives that:

$$\text{CoalescedTuples} = O\left(T \frac{d^{\log_C T}}{\log_C T!}\right) = O\left(T^{1/1+\log_d C}\right)$$

with the surprising result that even if we consider only two out of the three coalescings that dwarf performs, the size of the coalesced cube is only polynomial w.r.t to the dimensionality of the fact table and polynomial (and very close to linear) w.r.t to the number of tuples in the fact table.

<sup>4</sup>When we start creating the root node of the full dwarf there is no chance of left-coalescing, since nothing has been created

## 5 Algorithm for Dwarf Size Estimation

In this section we extend our analytical contribution to general case of varying cardinalities per dimensionality. Algorithm 1 can be used to estimate the number of coalesced tuples for sparse uniform data sets given the cardinalities of each dimension. We have extended the algorithm to work with zipfian distributions, but due to space constraints we refer the reader to the full version of the paper.

---

### Algorithm 1 SparsityTraverse Algorithm

---

**Input:** d: Number of Dimensions  
 Card: array of dimension cardinalities  
 FactT: current no of fact tuples  
 nc: tail coalesce flag(0 or 1)

- 1: **if** FactT=0 **then**
- 2:   return 0
- 3: **else if** FactT=1 **then**
- 4:   return nc {here tail or left-coalescing happens}
- 5: **else if** d=0 **then**
- 6:   return 1
- 7: **end if**
- 8: coalescedT  $\leftarrow$  0
- 9: mC  $\leftarrow$  Card[d]
- 10: zeroT  $\leftarrow$  mC  $\cdot e^{-\text{FactT}/\text{mC}}$
- 11: oneT  $\leftarrow$  FactT/(mC - 1)  $\cdot$  zeroT
- 12: **if** oneT  $\geq$  1 **then**
- 13:   x  $\leftarrow$  1
- 14:   **while** there are still fact tuples **do**
- 15:     xT  $\leftarrow$   $\binom{\text{FactT}}{x} / (\text{mC} - 1)^x \cdot \text{zeroT}$
- 16:     coalescedT       +=       SparsityTraverse(d-1, Card, xTuples, nc) {tail or left-coalescing may happen here}
- 17:     FactT -= xT
- 18:     x++
- 19:   **end while**
- 20: **else**
- 21:   coalescedT       +=       SparsityTraverse(d-1, Card, FactT/mC, nc) {drill-down traversal}
- 22: **end if**
- 23: coalescedT += SparsityTraverse(d-1, Card, FactT, 0) {roll-up traversal with left-coalescing}
- 24: return coalescedT

---

Initially the algorithm is called with the tail coalescing flag set to 1, since there is no chance for left-coalescing (there are no tuples to coalesce to). In line 4 we check if there is just one tuple in the subcube where tail or suffix

coalescing happens depending on the tail coalescing flag. In lines 12- 19 we traverse the basic partitioned node by checking iteratively how many cells get one, two, three, ... tuples until all the available tuples for the subcube are exhausted. The quantity:

$$\frac{\binom{\text{FactT}}{x}}{(\text{mC} - 1)^x} \cdot \text{mC} \cdot e^{-\text{FactT}/\text{mC}}$$

where FactT is the number of fact tuples for the current sub-dwarf and mC is the cardinality of the current dimension, returns the number of cells that get exactly x tuples<sup>5</sup>.

The algorithm works in a depth-first manner over the lattice and estimates recursively the number of coalesced tuples that its sub-dwarf generates. For example, for a three-dimensional cube *abc*, the algorithm in line 21 starts the *drill-down* to all subcubes with prefix *a* and recursively it proceeds to those with prefix *ab* and finally reaches prefixes *abc*, by estimating appropriately the number of tuples that each subdwarf gets. When (lines 1-7) there are no more dimensions to drill-down (or a tail or left coalescing can be identified), the drill-down over the subdwarfs with prefixes in *abc* stops and the algorithm *rolls-up* to the subdwarfs with prefixes *ab* in line 23 by setting the nc flag to 0 -since now there is possibility of left-coalescing with the subcubes in *abc*-. The process continues recursively to all the views of the lattice.

## 6 Related Work

The data cube operator is introduced in [6] and its potential has generated a flurry of research on a wide-variety of topics. Its exponential complexity on almost every aspect first guided to the rediscovery of materialized views and their adaptation. For example view selection algorithms can be found in [7, 8, 17]. However the general problem is show to be NP-Complete [10] and even greedy algorithms are polynomial in the number of views that need to consider which is actually exponential in the dimensionality of the datasets, rendering these approaches to a certain degree impractical for high-dimensional datasets.

Estimating the size of the data cube given its fact table is only addressed in [15] by using probabilistic techniques, however that approach cannot be extended to work with coalesced cubes.

<sup>5</sup>assuming a uniform distribution. For zipfian distribution we refer the reader to the full version of the paper



The problem of just computing the data cube appears especially interesting. Various techniques that try to benefit from commonalities between partitions or sorts, partial sorts and intermediate results are proposed in [2, 4, 14]. Other techniques that use multidimensional array representation techniques [20] suffer as well from the dimensionality curse. Techniques that try to exploit the inherent sparsity of the cube like [3, 12] seem to perform better.

Several indexing techniques have been devised for storing data cubes. Cube Forests [9], exploit prefix redundancy when storing the cube. In the Statistics Tree [5] prefix redundancy is partially exploited. Unique prefixes are stored just once, but the tree contains all possible paths (even non-existing paths) making it inappropriate for sparse datasets. Cubetrees[13] use packed R-trees to store individual views and exhibit very good update performance.

Recently compressed cubes are introduced which try to exploit the inherent redundancies in the structure of the cube. In [19] the notion of a *base single tuple* is introduced. Such a tuple is “shared” between different group-bys and is similar to the coalesced tuples discussed in this paper. However its applicability is limited since such tuples are discovered one at a time. Dwarf[16] provides a more efficient method for the automatic discovery of all type of suffix redundancies, since whole sub-cubes can coalesced *before* actually computing them. Dwarf also indexes the produced cube and is designed to work in secondary memory. QC-trees[11] use a bottom-up approach in discovering redundancies which checks if every grouping is redundant or not with every other grouping that it is possible to coalesce with.

## 7 Experiments

In this section we provide an extensive experimental evaluation of our approach based on synthetic and real data sets. We compare the results of our analytical approach with actual results taken from our implementation of Dwarf.

### 7.1 Synthetic Datasets

In Figure 10 we demonstrate how the number of coalesced tuples scales w.r.t to the dimensionality, for a uniform dataset. The number of fact table tuples was set to 100,000. We used two different cardinalities of 1,000 and 10,000. We see that our analytical approach provides extremely accurate results for large cardinalities. The reason that the error

decreases as the cardinality increase is the approximation in lemma 3, where we assume that  $C - 1 \approx C$ . The second observation has to do with the scalability w.r.t. to the dimensionality. The quantity  $\log_C T$  which determines the exponent of  $d$  is much smaller in the case of  $C = 10,000$  and therefore this data set scales better.

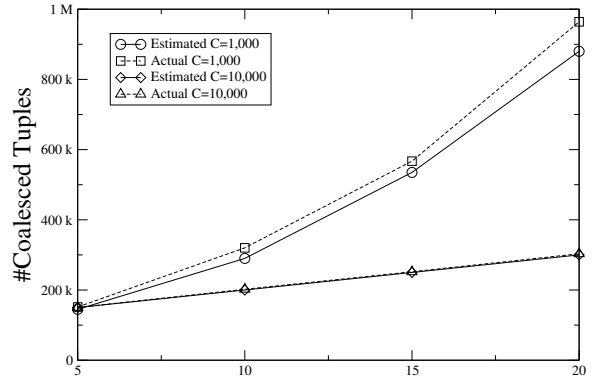


Figure 10: Accuracy v.s. dimensionality for varying cardinalities

### 7.2 Real Datasets

For this experiment we use a real eight-dimensional data set given to us by an OLAP company. The data set has varying cardinalities per dimension. We used various projections on the data set in order to decrease the dimensionality and study its effect on the accuracy. For this experiment the fact table had 672,771 tuples and two measures. Table 3 summarizes the parameters of each projection. Column “Projection” denotes the name of the data set, column  $d$  the number of dimensions and column “Cardinalities” the cardinalities of each dimension. In Figure 11 we depict the estimates of our approach compared with the actual numbers taken, when the dwarf is computed and stored.

| Projection | $d$ | Cardinalities                  |
|------------|-----|--------------------------------|
| A          | 5   | 1300,2307,2,2,3098             |
| B          | 6   | 1300,2307,3098,130,561,693     |
| C          | 7   | 1300,2307,2,3098,130,561,693   |
| D          | 8   | 1300,2307,2,2,3098,130,561,693 |

Table 3: Real data set parameters

We observe a very interesting pattern. As the dimensionality increases our approach *overestimates* increasingly more the coalesced size. The reason is that our approach currently handles *only sparsity coalescing* and ig-

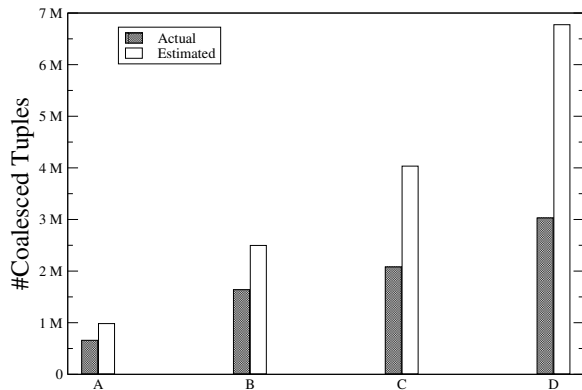


Figure 11: Accuracy v.s. dimensionality for real data set

nores the *implication coalescing* that is very apparent in high-dimensional data sets. As the dimensionality increases such implications increase and complement the sparsity implications reducing even further the coalesced size!

## 8 Conclusions

We have presented an analytical and algorithmic framework for estimating the size of coalesced cubes, where suffix redundancies diminish the number of aggregates that need to be stored and calculated. Our analysis on the Dwarf model although it uses only sparsity coalescing, derives the surprising result, that the coalesced cube grows polynomially w.r.t to the dimensionality! This result changes the establish state that the cube is exponential on the number of dimensions and extend the applicability of data warehousing methods to a much wider area. In addition we have demonstrated using real data, that the coalesced cube is *even smaller* than our analysis derives. The reason is that the effects of implication coalescing complement the results of sparsity coalescing that we have presented here.

## References

- [1] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional Samples for Approximate Answering of Group-By Queries. In *SIGMOD*, pages 487–498, Dallas, Texas, 2000.
- [2] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *VLDB*, pages 506–521, 1996.
- [3] K. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs. In *SIGMOD*, pages 359–370, Philadelphia, PA, USA, 1999.
- [4] P.M. Deshpande, S. Agarwal, J.F. Naughton, and R. Ramakrishnan. Computation of multidimensional aggregates. Technical Report 1314, University of Wisconsin - Madison, 1996.
- [5] Lixin Fu and Joachim Hammer. CUBIST: A New Algorithm for Improving the Performance of Ad-hoc OLAP Queries. In *DOLAP*, 2000.
- [6] J. Gray, A. Bosworth, A. Layman, and H. Piramish. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In *ICDE*, pages 152–159, New Orleans, February 1996. IEEE.
- [7] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index Selection for OLAP. In *ICDE*, pages 208–219, Birmingham, UK, April 1997.
- [8] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing Data Cubes Efficiently. In *SIGMOD*, pages 205–216, Montreal, Canada, June 1996.
- [9] T. Johnson and D. Shasha. Some Approaches to Index Design for Cube Forests. *Data Engineering Bulletin*, 20(1):27–35, March 1997.
- [10] H. J. Karloff and M. Mihail. On the Complexity of the View-Selection Problem. In *PODS*, pages 167–173, Philadelphia, Pennsylvania, May 1999.
- [11] L. Lakshmanan, J. Pei, and Yan Zhao. QC-Trees: An Efficient Summary Structure for Semantic OLAP. In *SIGMOD*, pages 64–75, San Diego, California, 2003.
- [12] K. A. Ross and D. Srivastana. Fast Computation of Sparse Datacubes. In *VLDB*, pages 116–125, Athens, Greece, 1997.
- [13] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos. Cube-tree: Organization of and Bulk Incremental Updates on the Data Cube. In *SIGMOD*, pages 89–99, Tucson, Arizona, May 1997.
- [14] S. Sarawagi, R. Agrawal, and A. Gupta. On computing the data cube. Technical Report RJ10026, IBM Almaden Research Center, San Jose, CA, 1996.
- [15] A. Shukla, P. Deshpande, J. Naughton, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *VLDB*, pages 522–531, Bombay, India, August 1996.
- [16] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. Dwarf: Shrinking the PetaCube. In *SIGMOD*, pages 464–475, Madison, Wisconsin, 2002.
- [17] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *VLDB*, pages 126–135, Athens, Greece, August 1997.
- [18] J.S Vitter, M. Wang, and B. Iyer. Data Cube Approximation and Histograms via Wavelets. In *CIKM*, 1998.
- [19] Wei Wang, Hongjun Lu, Jianlin Feng, and Jeffrey Xu Yu. Condensed Cube: An Effective Approach to Reducing Data Cube Size. In *ICDE*, 2002.
- [20] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *SIGMOD*, pages 159–170, 1997.

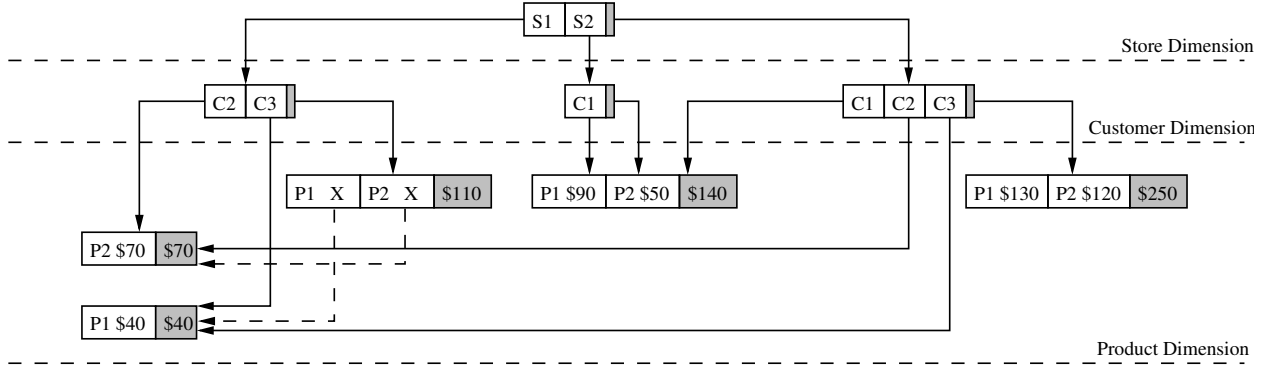


Figure 12: The Dwarf Cube for Table 1

## A Dwarf Representation

In order to visualize and define the coalescing properties we use the dwarf structure and describe how the structural redundancies appear under this model.

### A.1 Dwarf Structure

We briefly describe the Dwarf structure. The reader is directed to [16] for a complete discussion.

Figure 12 shows the Dwarf Cube for the fact table shown in Table 1. It is a full cube using the aggregate function *sum*. The height of the Dwarf is equal to the number of dimensions, each of which is mapped onto one of the levels shown in the figure. The root node contains cells of the form [key, pointer], one for each distinct value of the first dimension. The pointer of each cell points to the node below containing all the distinct values of the next dimension that are associated with the cell’s key. Each non-leaf node has a special ALL cell, shown as a small gray area to the right of the node, holding a pointer and corresponding to all the values of the node. Leaf nodes contain cells of the form [key, X], where X is either an aggregate or a pointer to another leaf node that contains the corresponding aggregate.

A path from the root to a leaf such as  $\langle S1, C3, P1 \rangle$  corresponds to an instance of the group-by (Store, Customer, Product) and leads to a cell [P1 \$40] which stores the aggregate of that instance. Some of the path cells can be open using the ALL cell. For example,  $\langle S2, ALL, P2 \rangle$  leads to the cell [P2 \$50], and corresponds to the sum of the Prices paid by any Customer for Product P2 at Store S2. At the leaf level, each cell is of the form [key, aggregate] and holds the aggregate of all tuples that match a path from the root to it. Each leaf node also has

an ALL cell that stores the aggregates for all the cells in the entire node.  $\langle ALL, ALL, ALL \rangle$  leads to the total Prices (group-by NONE).

Prefix redundancies are eliminated by storing each unique prefix just once, for example Store S1 is only stored just one in the DAG, as is the prefix S2, C1.

Suffix redundancies are visualized in nodes that are reachable through many paths. The reader can observe that the three paths  $\langle S2, C1, P2 \rangle$ ,  $\langle S2, ALL, P2 \rangle$ , and  $\langle ALL, C1, P2 \rangle$ , whose values are extracted from processing just the last tuple of the fact-table, all lead to the same cell [P2 \$50]. We refer to all nodes reachable through more than one path as *coalesced* nodes.