Research Report

Discovery-driven Exploration of OLAP Data Cubes

Sunita Sarawagi
Rakesh Agrawal
Nimrod Megiddo

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

IBM Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

Discovery-driven Exploration of OLAP Data Cubes

Sunita Sarawagi
Rakesh Agrawal
Nimrod Megiddo

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

ABSTRACT: Analysts predominantly use OLAP data cubes to identify regions of anomalies that may represent problem areas or new opportunities. The current OLAP systems support hypothesis-driven exploration of data cubes through operations such as drill-down, roll-up, and selection. Using these operations, an analyst navigates unaided through a huge search space looking at large number of values to spot exceptions. We propose a new discovery-driven exploration paradigm that mines the data for such exceptions and summarizes the exceptions at appropriate levels in advance. It then uses these exceptions to lead the analyst to interesting regions of the cube during navigation. We present the statistical foundation underlying our approach. We then discuss the computational issue of finding exceptions in data and making the process efficient on large multidimensional data bases. We present performance results and experience with real-life datasets to illustrate the effectiveness of the proposed paradigm.

1. Introduction

On-Line Analytical Processing (OLAP) characterizes the operations of summarizing, consolidating, viewing, applying formulae to, and synthesizing data along multiple dimensions. OLAP software helps analysts and managers gain insight into the performance of an enterprise through a wide variety of views of data organized to reflect the multidimensional nature of enterprise data [Col95]. An increasingly popular data model for OLAP applications is the multidimensional database [OLA96][AGS97], also known as the data cube [GBLP96]. A data cube consists of two kinds of attributes: *measures* and *dimensions*. The set of dimensions consists of attributes like product names and store names that together form a key. The measures are typically numeric attributes like sales volumes and profit. Dimensions usually have associated with them *hierarchies* that specify aggregation levels. For instance, *store name* $\rightarrow$ *city* $\rightarrow$ *state* is a hierarchy on the *store* dimension and *UPC code* $\rightarrow$ *type* $\rightarrow$ *category* is a hierarchy on the *product* dimension.

Hypothesis-driven Exploration. A business analyst while interactively exploring the OLAP data cube is often looking for regions of anomalies. These anomalies may lead to identification of problem areas or new opportunities. The exploration typically starts at the highest level of hierarchies of the cube dimension. Further, navigation of the cube is done using a sequence of "drill-down" (zooming in to more detailed levels of hierarchies), "roll-up" (zooming out to less detailed levels) and "selection" (choosing a subset of dimension members) operations. From the highest level of the hierarchy, the analyst drills-down to the lower levels of hierarchies by looking at the aggregated values and visually identifying interesting values to follow. Thus, drilling-down the product dimension from product category to product type may lead to product types whose sale exhibited some anomalous behavior. A further drill down may lead to individual product UPC codes causing this anomaly. If an exploration along a path does not lead to interesting results, the analyst rolls-up the path and starts pursuing another branch. A roll-up may lead to the top-level of hierarchy and then further drill-down may continue along another dimension.

This "hypothesis-driven" exploration for anomalies has several shortcomings. The search space is very large — typically, a cube has 5–8 dimensions, each dimension has a hierarchy that is 2–8 levels deep and each level of the hierarchy has ten to hundreds of members [Col95]. Simply looking at data aggregated at various levels of details to hunt down an anomaly that could be one of several million values hidden in detailed data is a daunting task. Furthermore, the higher level aggregations from where an analyst starts may not even be affected by an anomaly occuring underneath either because of cancellation of multiple exceptions or because of the large amount of data aggregated. Even if one is viewing data at the same level of detail as where the anomaly occurs, it might be hard to notice the exception because of large number of values.

Discovery-driven Exploration. We propose a new "discovery-driven" method of data exploration where an analyst's search for anomalies is guided by precomputed indicators of exceptions at various levels of detail in the cube. This increases the chances of user noticing abnormal patterns in the data at any level of aggregation. This paradigm could be especially valuable when the number of dimensions and hierarchies is large, making it overwhelming for the user to navigate through the multitudes of views of a data cube.

We present a formal notion of exceptions. Intuitively, we consider a value in a cell of a data cube to be an exception if it is significantly different from the value anticipated based on a statistical model. This model computes the anticipated value of a cell in context of its position in the data cube and combines trends along different dimensions that the cell belongs to. Thus, for instance, a large increase in sales in december might appear exceptional when looking at the time dimension but when looking at the other dimensions like product this increase will not appear exceptional if other products also had similar increase. The model allows exceptions to be found at all levels of aggregation.

We present computation techniques that make the process of finding exceptions efficient for large OLAP datasets. Our techniques use the same kind of data scan operations as required for cube aggregate computation [AAD+96] and thus enables overlap of exception finding with routine aggregate precomputation. These techniques recognize that the data may be too large to fit in main memory and intermediate results may have to be written to disk requiring careful optimization. We describe some experience of using this methodology on a real data set and give performance results.

Paper layout. The paper is organized as follows. In Section 2 we demonstrate a scenario of the use of our proposed method. Section 3 gives the statistical model we use to compute the anticipated value of a cell and the rationale for choosing this model. Computation techniques are discussed in Section 4. We conclude with a summary and directions for future work in Section 5.

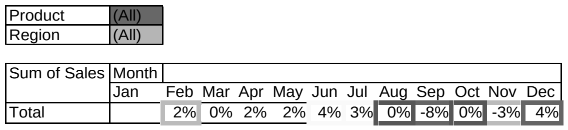
2. An Illustrative Example

We illustrate our proposed method using an example session with our prototype implementation. This prototype uses the Microsoft Excel spreadsheet, extended with appropriate macros, as the front-end for user-interaction. The backend is the well-known OLAP product, Arbor Essbase [Arb] that computes and stores the exceptions using the techniques we present in Sections 3 and 4.

To keep the example brief, we will consider a three-dimensional data cube with dimensions Product, Market, and Time. There is a hierarchy Market $\rightarrow$ Region $\rightarrow$ ALL on the Market dimension. The data for this cube is taken from a sample OLAP database distributed with Essbase [Arb].

We annotate every cell in all possible aggregations of a data cube with a value that indicates the degree of "surprise" that the quantity in the cell holds. The surprise value captures how anomalous a quantity in a cell is with respect to other cells. The surprise value of a cell is a composite of the following three values (we give definitions and discuss how these values are determined in Section 3):

1. $\mathcal{S}$elfExp: represents the surprise value of the cell relative to other cells at the same



Product	(All)
Region	(All)

Sum of Sales	Month											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Total		2%	0%	2%	2%	4%	3%	0%	-8%	0%	-3%	4%

Figure 1: Change in sales over time

level of aggregation.

2. $\mathcal{I}$nExp: represents the degree of surprise somewhere beneath this cell if we drill down from the cell.

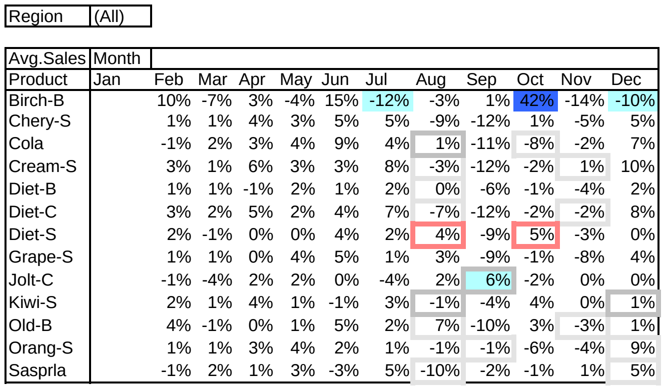
3. $\mathcal{P}$athExp: represents the degree of surprise for each drill-down path from the cell.

Consider a user looking at the monthly sales as a percentage difference from the previous month. Suppose the user starts by viewing the data aggregated over all products and markets for different months of the year as shown in Figure 1.

To find out what parts of the cube may be worthy of exploring further in terms of exceptions, the user invokes a "`highlight exceptions`" button that colors the background of each cell based on its $\mathcal{S}$elfExp value. In addition, each cell is surrounded with a different colored box based on the $\mathcal{I}$nExp value. In both cases, the intensity of the color is varied with the degree of exception. In Figure 1, the months with a thick box around them have a high $\mathcal{I}$nExp value and thus need to be drilled down further for exceptions underneath them. Darker boxes (e.g., around "Aug", "Sep" and "Oct") indicate higher values of $\mathcal{I}$nExp than the lighter boxes (e.g., around "Feb" and "Nov").

There are two paths the user may drill down along from here: Product and Region. To evaluate which of these paths has more exceptions, the user selects a cell of interest and invokes a "`path exception`" module that colors each aggregated dimension based on the surprise value along that path. These are based on the $\mathcal{P}$athExp values of the cell. In Figure 1 (top-left part) the path along dimension Product has more surprise than along Region indicated by darker color. Drilling-down along Product yields 143 different sales values corresponding to different Product-Time combinations as shown in Figure 2. Instead of trying to find the exceptions by manual inspection, the user can click on the "`highlight exception`" button to quickly identify the exceptional values. In this figure, there are a few cells with high $\mathcal{S}$elfExp values and these appear as cells with a different background shade than the normal ones (darker shades indicate higher surprise). For instance, sales of "Birch-B(eer)" shows an exceptional difference of 42% in the month of "Oct". In addition, three other cells are also indicated to have large $\mathcal{S}$elfExp values although the sales values themselves (6% for <Jolt-C, Sep>, -12% for <Birch-B, Jul>and -10% for <Birch-B, Dec>) are not exceptionally large when compared with all the other cells. The reason why these cells are marked as exceptions will be explained in Section 3.

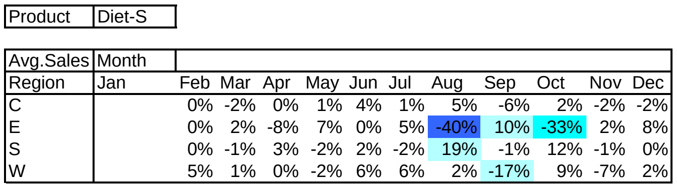
Figure 2 also shows some cells with large $\mathcal{I}$nExp values as indicated by the thick boxes around them. The highest $\mathcal{I}$nExp values are for Product "Diet-S(oda)" in the months of "Aug" and "Oct". The user may therefore choose to explore further details for "Diet-Soda" by drilling down along Region. Figure 3 shows the sales figures for "Diet-Soda" in



Region	(All)

Avg.Sales	Month											
Product	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Birch-B		10%	-7%	3%	-4%	15%	-12%	-3%	1%	42%	-14%	-10%
Chery-S		1%	1%	4%	3%	5%	5%	-9%	-12%	1%	-5%	5%
Cola		-1%	2%	3%	4%	9%	4%	1%	-11%	-8%	-2%	7%
Cream-S		3%	1%	6%	3%	3%	8%	-3%	-12%	-2%	1%	10%
Diet-B		1%	1%	-1%	2%	1%	2%	0%	-6%	-1%	-4%	2%
Diet-C		3%	2%	5%	2%	4%	7%	-7%	-12%	-2%	-2%	8%
Diet-S		2%	-1%	0%	0%	4%	2%	4%	-9%	5%	-3%	0%
Grape-S		1%	1%	0%	4%	5%	1%	3%	-9%	-1%	-8%	4%
Jolt-C		-1%	-4%	2%	2%	0%	-4%	2%	6%	-2%	0%	0%
Kiwi-S		2%	1%	4%	1%	-1%	3%	-1%	-4%	4%	0%	1%
Old-B		4%	-1%	0%	1%	5%	2%	7%	-10%	3%	-3%	1%
Orang-S		1%	1%	3%	4%	2%	1%	-1%	-1%	-6%	-4%	9%
Sasprla		-1%	2%	1%	3%	-3%	5%	-10%	-2%	-1%	1%	5%

Figure 2: Change in sales over time for each product



Product	Diet-S

Avg.Sales	Month											
Region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
C		0%	-2%	0%	1%	4%	1%	5%	-6%	2%	-2%	-2%
E		0%	2%	-8%	7%	0%	5%	-40%	10%	-33%	2%	8%
S		0%	-1%	3%	-2%	2%	-2%	19%	-1%	12%	-1%	0%
W		5%	1%	0%	-2%	6%	6%	2%	-17%	9%	-7%	2%

Figure 3: Change in sales of Product "Diet-Soda" over time in each Region

different Regions. By highlighting exceptions in this plane, the user notices that in Region "E" (for Eastern), the sales of "Diet-Soda" has decreased by an exceptionally high value of 40% and 33% in the months of "Aug" and "Oct" respectively. Notice that the sales of "Diet-Soda" in the Product-Time plane aggregated over different Regions (Figure 2) gives little indication of these high exceptions in the Region-Product-Time space. This shows how the $\mathcal{I}$nExp value at higher level cells may be valuable in reaching at exceptions in lower level cells.

There are no other cells with high $\mathcal{I}$nExp in Figure 3. Therefore, the user may stop drilling down and go back to the Product-Time plane of Figure 2 to explore other cells with high $\mathcal{I}$nExp. Suppose, he chooses to drill-down along Product "Cola" in "Aug". Figure 4 shows the exceptions for "Cola" after drilling down along Region. The "Central" Region has a large $\mathcal{I}$nExp and may be drilled down further, revealing the $\mathcal{S}$elfExp values in the Market-time plane for "Cola". (as shown in Figure 5).

Market	(All)
Product	Cola

Avg.Sales	Month										
Region	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
C		3%	1%	4%	1%	4%	10%	-11%	-14%	-3%	5%	11%
E		-3%	3%	4%	4%	13%	2%	0%	-10%	-13%	-3%	8%
S		2%	-1%	1%	9%	6%	3%	21%	-15%	1%	-5%	4%
W		-2%	2%	2%	4%	12%	1%	1%	-9%	-11%	-4%	6%

Figure 4: Change in sales over Time for Product “Cola” in different Region

Product	Cola
Region	C

Avg.Sales	Month											
Market	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Colorado		0%	2%	-8%	7%	0%	5%	-40%	10%	-32%	2%	8%
Illinois		3%	4%	24%	2%	3%	19%	-18%	-17%	-5%	12%	30%
Iowa		2%	0%	-3%	11%	3%	7%	20%	-20%	-6%	-6%	8%
Missouri		2%	-2%	-6%	-3%	-2%	0%	-5%	-6%	18%	4%	-17%
Ohio		-9%	-5%	-12%	-4%	-10%	-9%	0%	26%	0%	7%	-11%
Wisconsin		16%	3%	0%	-3%	19%	13%	7%	-30%	3%	-3%	12%

Figure 5: Change in sales over Time for Product “Cola” in the “Central” Region

Product	Birch-B
Region	E

Sum of Sales	Month											
State	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Massachusetts		0%	-4%	0%	-8%	0%	-2%	0%	7%	35%	-14%	-16%
New-Hampshire		5%	10%	-13%	17%	3%	14%	-27%	-17%	57%	-11%	-41%
New-York		18%	-10%	8%	-4%	24%	-19%	-1%	1%	44%	-15%	-3%

Figure 6: Change in sales over Time for Product “Birch-B”

3. Defining exceptions

Intuitively, a value in a cell of a data cube is an exception if it is surprising. There could be several interpretations of this notion. We present the approach we use. Later in Section 3.8 we discuss the alternatives we considered before deciding on our approach.

Our choice of exception model was motivated by the following desiderata:

1. We need to consider variation and patterns in the measure value across all dimensions that a cell belongs to. This helps us find values that are exceptional within the context of a particular aggregation. It is not enough to simply treat the values in a cube as a flat set and call extreme values in the set as exceptions. For instance, consider the example data cube from Figure 2. If we ignored cell positions, possibly only <Birch-B, Oct> would be marked as an exception. However, we identify several other exceptions. Interestingly, the entry <Birch-B, Dec> with value -10% is marked as an exception whereas entry <Birch-B, Nov> with a higher value -14% is not because for the "Dec" column almost all other product have a large positive value whereas in the "Nov" column most other products also have a negative value. In the "Sep" column, "Jolt-C" has a relatively large positive value since most other products have a negative value and is therefore marked as an exception.

2. We need to find exceptions at all aggregated group-bys of the cube, and not only at the detailed level because it simplifies end-user comprehensibility through concise representation. For instance, $\langle$Birch-B,Oct$\rangle$ is an exception at the $\langle$Product,Month$\rangle$ group-by (Figure 2) and that means we do not need to mark as exceptions all the high values for $\langle$Birch-B,Oct, *$\rangle$ at the $\langle$Product,Month,State$\rangle$ group-by (Figure 6).

3. The user should be able to interpret the reason why certain values are marked as exceptions. A typical OLAP user is a business executive, not necessarily a sophisticated statistician. We are targeting our system for this audience. Our method therefore should not require the user to make choices between complex statistical models and the process of finding exceptions should be fairly automated.

4. The procedure for finding exceptions should be computationally efficient and scale for large datasets commonly found in OLAP databases. Also, it should be generalizable and efficient for the range of dimensions that are common in OLAP (typically 3 to 8) and handle hierarchies on dimensions.

3.1. The Model

Consider first the problem of finding exceptions in the most detailed values of the data cube. We call a value an exception if it differs significantly from the anticipated value calculated using a model that takes into account all aggregates (group-bys) in which the value participates. This model was inspired by the table analysis methods [HMJ88] used in the statistical literature.

For a value $y_{i_1 i_2 \ldots i_n}$ in a cube C at position i_r of the rth dimension d_r $(1 \leq r \leq n)$, we define the anticipated value $\hat{y}_{i_1 i_2 \ldots i_n}$ as a function f of contributions from various higher level group-bys as:

$$\hat{y}_{i_1 i_2 \ldots i_n} = f(\gamma^{G}_{(i_r | d_r \in G)} | G \subset \{d_1, d_2, \ldots d_n\}) \tag{3.1}$$

We will refer to the γ terms as the *coefficients* of the model equation. The way these coefficients are derived is explained in Section 3.4. The different functional forms function f can take is discussed in Section 3.3.

We clarify Eq. 3.1 by illustrating for the case of a cube with three dimensions A, B, C. The anticipated value $\hat{y}_{ijk}$ for the ith member of dimension A, jth member of dimension B and kth member of dimension C, is expressed as a function of seven terms obtained from each of the seven group-bys of the cube as:

$$\hat{y}_{ijk} = f(\gamma, \gamma^A_i, \gamma^B_j, \gamma^C_k, \gamma^{AB}_{ij}, \gamma^{BC}_{jk}, \gamma^{AC}_{ik})$$

The absolute difference between the actual value, $y_{i_1 i_2 \ldots i_n}$ and the anticipated value $\hat{y}_{i_1 i_2 \ldots i_n}$ is termed as the residual $r_{i_1 i_2 \ldots i_n}$ of the model. Thus,

$$r_{i_1 i_2 \ldots i_n} = |y_{i_1 i_2 \ldots i_n} - \hat{y}_{i_1 i_2 \ldots i_n}| \ .$$

Intuitively, any value with a *relatively* large value of the residual is an exception. A statistically valid definition of "relatively large" requires us to scale the values based also on the anticipated standard deviation $\sigma_{i_1 i_2 \ldots i_n}$ associated with the residuals. Thus, we call a value an exception if the standardized residual, $s_{i_1 i_2 \ldots i_n}$, defined as

$$s_{i_1 i_2 \ldots i_n} = \frac{|y_{i_1 i_2 \ldots i_n} - \hat{y}_{i_1 i_2 \ldots i_n}|}{\sigma_{i_1 i_2 \ldots i_n}} \tag{3.2}$$

is higher than some threshold τ. We use $\tau = 2.5$ corresponding to a probability of 99% in the normal distribution. In Section 3.5 we discuss how we estimate the standard deviations.

3.2. Exceptions at Higher Levels of Group-bys

Exceptions at higher level group-bys of the cube can be found by separately fitting the model Eq. 3.1 on aggregated values at each group-by of the data cube using different values of n. For instance, for a cube with three dimensions A, B, C we will need one equation at the most detailed level ABC where $n = 3$, three equations for group-bys AB, BC and CA where $n = 2$, and three equations for group-bys A, B and C where $n = 1$. The OLAP user specifies the aggregate function to be used for summarizing values at higher levels of the cube. For instance, a user might specify "sum" or "average" of sales as the aggregate function. Accordingly, exceptions in "total" or "average" sales will be reported at various group-bys of the cube.

3.3. Functional forms of f

The function f in Eq. 3.1 can take a form which is:

- Additive: the function f returns the sum of its arguments.
- Multiplicative: the function f returns the product of its arguments.

Other (more complex) functional forms for f are also possible — most of them involving different mixtures of additive and multiplicative terms [HMJ88]. A significantly different approach in this category is the one suggested in [Man71] where factor analytic models like the singular value decomposition [CL86] are used to fit a model based on a mixture of additive and multiplicative terms. The main demerit of these models is the high overhead of computing them and the lack of generalizations of the models to more than 2-3 dimensions and hierarchies.

In our experience with OLAP datasets, the multiplicative form provided better fit than the additive form. For ease of calculation, we transform the multiplicative form to a linear additive form by taking a log of original data values. We thus have

$$\hat{l}_{i_1 i_2 \ldots i_n} = \log \hat{y}_{i_1 i_2 \ldots i_n} = \sum_{G \subset \{d_1, d_2, \ldots d_n\}} \gamma^{G}_{(i_r | d_r \in G)} \tag{3.3}$$

For a three-dimensional cube, this equation takes the form:

$$\hat{l}_{ijk} = \log \hat{y}_{ijk} = \gamma + \gamma^{A}_{i} + \gamma^{B}_{j} + \gamma^{C}_{k} + \gamma^{AB}_{ij} + \gamma^{BC}_{jk} + \gamma^{AC}_{ik}.$$

To get a intuition for why the multiplicative form should fit the OLAP data well, consider a two-dimensional cube with dimensions “product” and “store” where the values in the cell indicate counts of total units sold. This cube is generated by counting how many unit transactions correspond to a particular store and product. Let y_{++} be the total number of unit transactions and y_{ij} denote the count for product i and store j. We can model the transactional data as (an empirical) multinomial distribution with variables store and product. The joint probability distribution of these two variables is represented by the values in the cube when we divide each value by y_{++}, i.e., $p_{ij} = y_{ij}/y_{++}$. Let p_{i+} denote the marginal probability of a transaction on product i and let p_{+j} denote the probability of it from store j. When the variables are independent, the joint probability $p_{ij} = p_{i+}p_{+j}$. Thus, $y_{ij} = y_{++}p_{i+}p_{+j} = \frac{y_{i+}y_{+j}}{y_{++}}$. Thus, the contribution of row i and column j are *multiplied* together. This is an approximation of the complex real world phenomenon that generated the data. But, it helps us form statistical justification for the empirically observed superior fits for the multiplicative model.

3.4. Estimating model coefficients

We now discuss how we estimate the coefficients of the model equation. Two possible approaches are:

1. Mean-based estimates: For deriving these estimates we assume the logarithms of the values are distributed normally with the same variance. The following approach yields the least-squares estimates in that case [HMT83]:

   - $\gamma = \ell_{+\ldots+}$ which is the grand mean or average. Note that a "+" in the ith index denotes an aggregation along the ith dimension.
   - $\gamma_{i_r}^{A_r} = \ell_{+..+i_r+..+} - \gamma$ where $\ell_{+..+i_r+..+}$ is the mean over all values along i_rth member of dimension A_r. Thus, $\gamma_{i_r}^{A_r}$ denotes how much the average of the values along i_rth member of dimension A_r differs from the overall average.
   - $(\gamma)_{i_r i_s}^{A_r A_s} = \ell_{+..+i_r+..+i_s+..+} - \gamma_{i_r}^{A_r} - \gamma_{i_s}^{A_s} - \gamma$.

   In general, the coefficients corresponding to any group-by G are obtained by subtracting from the average ℓ value at group-by G all the coefficients from higher level group-bys. Intuitively, the coefficients reflect an adjustments to the mean of the corresponding group-by after all higher-level adjustments are taken into account. If a user is navigating the data cube top-down, then the coefficients reflect how different the values at more detailed levels are, based on the general impressions formed by looking at higher level aggregates. This helps provide easy grasp of why certain numbers are marked exceptions.

2. Other robust estimates: The main shortcoming of the mean-based approach is that it is not robust in the presence of extremely large outliers. Therefore, a number of methods including the median polish method [HMJ88] and the square combining method [HMJ88] have been proposed. These are all based on using robust estimates of central tendency like "median" or "trimmed-mean" instead of "mean" for calculating the coefficients. Trimmed-mean of a set of values is defined as the mean of the values left after a certain fraction of the extreme values (largest and smallest) have been trimmed off.

We used the 75% trimmed-mean where 25% of the extreme values are trimmed off and the mean is taken of the middle 75% numbers. By dropping 25% of the extreme numbers, we make the method robust to outliers.

3.5. Estimating standard deviation

In classical Analysis of Variance (ANOVA) methods [Mon91], the standard deviation for all the cells is assumed to be identical. The variance (square of standard deviation) is estimated as the sum of squares of the residuals divided by the number of entries. We found that this method provides poor fits on OLAP data. In the analysis of contingency tables [BFH75], where cell entries represent counts, the Poisson distribution is assumed. This assumption implies that the variance is equal to the mean. When the entries are not counts (e.g., large dollar values), this typically leads to an underestimate of the variance.

The method we use for estimating variance is based on a slight modification of the previous models. We model the variance as a power ρ of the mean value $\hat{y}_{i_1\ldots i_n}$ as:

$$\sigma^2_{i_1 i_2 \ldots i_n} = (\hat{y}_{i_1 i_2 \ldots i_n})^{\rho} \ .$$

To calculate ρ we use the maximum likelihood principle [CL86] on data assumed to be distributed normally with the mean value $\hat{y}_{i_1 i_2 \ldots i_n}$. According to the latter, one can derive that the estimated value of ρ must satisfy:

$$\sum \frac{(y_{i_1 i_2 \ldots i_n} - \hat{y}_{i_1 i_2 \ldots i_n})^2}{(\hat{y}_{i_1 i_2 \ldots i_n})^{\rho}} \cdot \log \hat{y}_{i_1 i_2 \ldots i_n} - \sum \log \hat{y}_{i_1 i_2 \ldots i_n} = 0 \ . \tag{3.4}$$

The method we used for solving the equation to find ρ is discussed Section 4.

3.6. Summarizing exceptions

As discussed in Section 2, we need to summarize exceptions in lower levels of the cube as single values at higher levels of cube. For all cells displayed, the user should be able to find out: (1) what cells are exceptions in their respective group-bys; (2) what cells need to be drilled down further for exceptions underneath them and, for each such cell; (3) what is the best path for getting to the exceptions. These questions are answered by the three kinds of quantities: $\mathcal{S}$elfExp, $\mathcal{I}$nExp and $\mathcal{P}$athExp that we introduced in Section 2. We now define them in terms of the model coefficients.

$\mathcal{S}$elfExp:. denotes the exception value of the cell. This quantity is defined as the scaled absolute value of the residual defined in Eq. 3.2 with a cut-off threshold of τ. Formally,

$$\mathcal{S}\text{elfExp}(y_{i_1 i_2 \ldots i_n}) = \max\left(\frac{|y_{i_1 i_2 \ldots i_n} - \hat{y}_{i_1 i_2 \ldots i_n}|}{\sigma_{i_1 i_2 \ldots i_n}} - \tau, 0\right)$$

$\mathcal{I}$nExp:. denotes the total degree of surprise over *all* elements reachable by drill-downs from this cell. One definition of $\mathcal{I}$nExp could be the *sum* of the $\mathcal{S}$elfExp of all cells underneath it. However, with sum, several small $\mathcal{S}$elfExp could add up to give a larger value of $\mathcal{I}$nExp than a cell containing a few but very large $\mathcal{S}$elfExps underneath it. The "Max" function seemed to be a better choice for our purpose. Formally, the $\mathcal{I}$nExp of a cell $y_{i_1 i_2 \ldots i_n}$:

$$\mathcal{I}\text{nExp}(y_{i_1 i_2 \ldots i_n}) = \max\{\mathcal{S}\text{elfExp}(y_{j_1 j_2 \ldots j_n}) | (\forall r, 1 \le r \le n, i_r = j_r \ \textit{or} \ i_r = +) \ \& \ (\{j_1, \ldots j_n\} \ne \{i_1, \ldots i_n\})\}$$

$\mathcal{P}$athExp:. denotes the degree of surprise to be anticipated if drilled down along a particular path for each possible drill down path from the cell. We define $\mathcal{P}$athExp as the maximum of the $\mathcal{S}$elfExp over all cells reachable by drilling down along that path. Formally,

$$\mathcal{P}\text{athExp}(y_{i_1 i_2 \ldots i_n}, k) = \max\{\mathcal{S}\text{elfExp}(y_{j_1 j_2 \ldots j_n}) | (\forall r, 1 \le r \le n, i_r = j_r \ \textit{or} \ i_r = +) \ \& \ j_k \ne +\}, \forall k \text{ where } i_k = +$$

		Products					All	
	Stores	1	2	3	4	5	$\mathcal{S}$elfExp	$\mathcal{I}$nExp
	1	0	0	0	3	0	0	3
	2	0	3	0	0	0	3.5	3
	3	0	3	0	0	0	0	3
	4	0	3	0	0	0	0	3
	5	0	0	0	0	0	2.8	0
All	$\mathcal{S}$elfExp	0	0	0	0	0	0	-
	$\mathcal{I}$nExp	0	3	0	3	0	-	3.5

Table 1: Residuals in the product store plane

Example. Table 1 illustrates these definitions with a hypothetical example of a Product-Store cube showing the $\mathcal{S}$elfExp values in the innermost square of cells. For cells in the Product-Store plane only $\mathcal{S}$elfExp values are meaningful, whereas for the Product plane (bottom rectangle) or the Store plane (rightmost rectangle) both the $\mathcal{S}$elfExp and $\mathcal{I}$nExp values are shown. For the ALL group-by (shown in the bottom-rightmost rectangle), the $\mathcal{I}$nExp is the maximum over the $\mathcal{I}$nExp and $\mathcal{S}$elfExp of the product group-by and store group-by. The $\mathcal{P}$athExp is also of interest for this group-by since there are two drill-down paths from it. Path store has a $\mathcal{P}$athExp value of 3.5 whereas the product path only has a value of 3.

3.7. Other extensions

3.7.1. Hierarchies. Our model equation (Eq. 3.3) used for calculating the expected value can be extended to handle hierarchies along one or more dimensions of the cube. The basic idea adapted from [Mon91] is to define the anticipated value, not only based on row and column position but also on its parents along the hierarchies. For instance, consider values ℓ_{ij} in a cube consisting of two dimensions A and B where dimension A has two levels of hierarchies: $A^1 \rightarrow A^2 \rightarrow ALL$. To calculate an anticipated value at A^1B group-by, in addition to the row coefficient $\gamma_i^{A^1}$ at group-by A^1, column coefficient γ_j^B at group-by B and overall coefficient γ at group-by ALL, we have two new terms corresponding to the two new aggregations A^2 and A^2B along the hierarchy on A. The modified equation is:

$$\hat{\ell}_{ij} = \gamma + \gamma_i^{A^1} + \gamma_j^B + \gamma_{i'}^{A^2} + \gamma_{i'j}^{A^2B}$$

where i' denotes the parent of i at hierarchy level A^2, $\gamma_{i'}^{A^2}$ denotes the contribution of the i'th value at group-by A^2 and $\gamma_{i'j}^{A^2B}$ denotes the contribution due to the $i'j$th value at group-by A^2B. These additional coefficients have the effect of conceptually dividing the original two-way table on A and B into a collection of sub-tables on A^2B corresponding to different values of A^1. This is justified since we expect values belonging to the same group of a hierarchy to behave similarly.

The general formula for handling hierarchies is to express a value in a cube in terms of coefficients obtained from all of its less detailed group-bys. For instance, for the values at A^1B in the equation above, we used coefficients from the five higher group-bys:

A^1, A^2, B, A^2B, ALL. For estimating these coefficients we follow the same recursive rule of Section 3.4 where we first estimate γ as the overall average and then for terms corresponding to each group-by G we compute the average at G and then subtract from it the coefficients from each child of G.

3.7.2. Time series regression. Data in the OLAP cube often has a time dimension. Time is special in that it is ordered unlike nominal dimensions such as product and store. Therefore, one expects to find patterns like trends (steady increase or decrease in mean value over time) and seasonality (cyclic changes at periodic intervals) [Cha84] that are specific to time.

The model equation is very powerful in that it can automatically filter out trends and seasonality common to all elements of the cube even when treating time as a categorical attribute. For instance, suppose the total sales of *all* stores is much higher in December than in any other month. When looking at the sales of a single store "s" along time, the sales in December might appear exceptionally high compared to other months but when comparing the sales in December across all stores, the high value of sales for store s in December will not appear exceptional. Only, the coefficient corresponding to the month of December every year will have a high value. These exceptions can also by filtered away by defining a hierarchy on months of the year. Thus, all "December" months will belong to the same node of the hierarchy and the coefficient in the model equation corresponding to "December" will have a high value. Therefore, we will not report December of each of the five years as an exception. We get only one exception in the month of December over all years. Thus, appropriate hierarchies can be defined to account for seasonality in the time dimension.

More extensive and refined modeling is also possible to automatically detect trends and seasonality in the data using other methods of time series analysis [Cha84]. For instance, we can extend the model equation with additional terms contributed from adjacent points in time. We defer this discussion to future.

3.8. Alternative approaches

We chose the exception model described above after studying several alternatives. We discuss some of the ones considered to justify our selection.

Extreme values in a set:. The simplest approach would be to treat all the values in the cube as a flat set and mark any extreme value as an exception. Several methods of varying complexity can be used for identifying extreme values in a flat set of values:

- Values that are more than a certain standard deviations away from the mean [HMT83].
- Elements that cause the largest decrease in variance per element removed [AAR96].

Clustering:. In this method, one first clusters the values and then marks as exceptions values that do not belong to any of the major clusters. Several existing clustering techniques [JD88] could be used for this purpose after appropriate modifications. (Most of them discard exceptions as noise [SD90]; we, in contrast, want to isolate small minorities).

The shortcoming of both of the above methods is that in marking a value as an exception we are not taking into account the position and context in which the values occurs.

Multi-dimensional clustering:. In this method one treats each point as an $n + 1$ dimensional point (consisting of the n dimensions and the measure value) and finds clusters in this $n+1$ dimensional space. This way one takes into account the entire context in which the point occurs.

The problem with this method is that for categorical dimensions (like products and stores) the notion of clusters is often not meaningful and in OLAP data cubes the dimensions are frequently categorical.

Regression on continuous dimensions:. In the rare cases where the cube only contains continuous valued dimensions like age and salary, one can use the rich statistics literature on outlier diagnostics for multivariate data [RL87, Joh92]. Most of these are based on fitting regression models on the data and then isolating as exceptions data values that have a large influence on the regression estimator.

Extreme values in multiple groups with same categorical value:. In this method we keep one set of categorical dimensions fixed and values corresponding to the same value of these dimensions are put in the same group. In each of these groups we find exceptions using an extreme-value finding algorithm discussed earlier.

For a cube with attributes A,B,C,D, for each group-by pair (X, Y) where X is a subset of Y, e.g., (AB, ABCD), one can find exceptions in ABCD by grouping numbers with same value of AB. Thus, each group-by will be associated with a set of exceptions found with respect to different groupings of it, e.g., for ABCD one needs to find exceptions with respect to ABC, ABD, AB, A and so on. This is the approach used by the Explora project [Klo95] where a number is marked *interesting* or exceptional when it is significantly different from the average value amongst numbers in any larger group to which it belongs.

The problem with this approach is that one may get a large number of exceptions and most of these could be quite meaningless when inspected in the context of multiple overlapping grouping regions.

Combined effects of categorical dimensions:. Instead of treating each group of values separately, one develops an expectation of the value by simultaneously looking at the value in the context of all its dimensions. Our method falls under this category. There are several ways in which one can combine the effect of different dimensions based on the different functional forms and the different ways of estimating the model parameters. We have already discussed these alternatives and the rationale for our choice in sections 3.3, 3.4 and 3.5.

4. Computation Techniques

At first glance, our approach may appear unrealizable in practice because of the apparent high cost of computing exceptions at every cell of every group-by of the cube. In this section, we present fast computation techniques that make our approach feasible for large OLAP databases. There are three logical phases in the computation of exceptions in the entire cube:

1. The first phase involves the computation of the aggregate values (as specified by the user-provided aggregate function) over which exceptions will be found at each group-by of the cube. This is essentially the problem of cube computation and efficient computation techniques for this problem have been developed in [AAD+96].

2. The next phase is *model fitting*, *i.e.*, finding the coefficients of the model equation and using them to find the residuals as discussed in Section 3.1.

3. The final phase involves summarizing exceptions found in the second phase as discussed in Section 3.6. Computationally, this phase is similar to phase 1. There is, however, one difference. Each group-by needs to aggregate *all* of its immediate parents for calculating the $\mathcal{P}$athExp values whereas for phase 1 *any one* of the parent group-by can be used for computing the aggregate functions. Therefore, the optimizations based on choosing a good parent for each group-by [AAD+96] are inapplicable here. However, other optimizations [AAD+96] based on pipelining the computation of multiple group-bys and using partially sorted parents are applicable. Also, we need to consider only tuples that have non-zero $\mathcal{S}$elfExp and $\mathcal{I}$nExp values. Thus, this phase can be expected to be much faster than the other phases.

In the rest of this section, we will focus on phase 2. The computation needs to be planned carefully because in most cases, the data will be too large to fit in main memory and intermediate results will often have to be read/written to disk requiring careful optimizations.

4.1. Model fitting

In general, we need to fit separate equations for different group-bys of the cube as discussed in Section 3.2. We will first consider the scenario where a single equation is fit on the base level data. Later in Section 4.2, we will discuss how to simultaneously fit multiple equations, one for each of the group-bys of the cube.

We first present a method called **UpDown** that is directly based on Eq. 3.3 and later present improvements.

4.1.1. The UpDown Method. Recall from Section 3.4 that the coefficients at each group-by G of the cube is equal to the average value at G minus the sum of the coefficients of all group-bys that are subsets of G. Thus, an efficient way to compute the coefficients is the following two pass approach: First in the `up-phase`, compute the average ℓ value (call it `avg-l`) at each group-by starting from the most detailed group-by. This is computationally similar to the cube computation of phase 1 where we compute the user specified aggregate function (call it `user-agg`). Thus, phase-1 and the `up-phase` of phase 2 can be combined to save on the disk scan and sorting costs. Then in the `down-phase`, subtract from each group-by G the coefficients of all its subsets starting from the least detailed group-by (ALL).

Find-coefficients

```
Up-phase:
    For each group-by $G$ starting from the most detailed group-by
        Compute the user-agg and avg-l values from one of its parents
Down-phase:
    For each group-by $G$ starting from the least detailed
        Compute coefficient at $G$ by subtracting from avg-l values, coefficients
            from all group-bys $H$ where $H \subset G$.
```

Example:. Consider a three attribute cube ABC. We first compute the average value for each of the $2^3 - 1 = 7$ group-bys of the cube by starting from the ABC group-by and computing the average at AB, AC and BC from ABC, computing the average at A from one of AB or AC and so on, using the cube computation methods of [AAD+96]. We then compute the coefficient starting from ALL. The coefficient of each member of group-by A is the average value at the member minus the coefficient of its parent ALL, the coefficients at AB is the average at AB minus the coefficients at A, B and ALL and so on. Finally, we subtract from the average ℓ value at ABC coefficients at AB, AC, BC, A, B, C and ALL.

Analysis. The `up-phase` involves cube aggregation as in phase 1. The `down-phase` is computationally rather intensive because, in general, for computing the coefficients of a n attribute group-by we need to subtract coefficients from $2^n - 1$ other group-bys. This is equivalent to joining the n-attribute group-by with $2^n - 1$ other group-bys. When the size of these group-bys is large, computing so many multi-attribute joins per group-by can incur large sorting and comparison costs.

4.1.2. Modified UpDown method. We can reduce the number of sorts and I/O costs needed for the `down-phase` by subtracting together from each group-by G, a collection of its subset group-bys instead of subtracting one subset at a time. Choosing which subsets to collect together requires careful planning. To efficiently perform the subtraction operation between a group-by G and its subset H we sort both G and H in the same order. A group-by G (on more than one attribute) cannot simultaneously be sorted with respect to all of its subsets. Therefore, we partition the subset group-bys such that G is sorted once for each group-by in the same partition. Minimizing the number of such partitions leads to minimization of sorting and I/O costs. We present below a greedy procedure for determining the partitions and performing the subtract operation on each group-by. (We also have an algorithm for finding the optimal number of partitions using a maximal-matching based algorithm developed using ideas similar to those in [AAD+96]. However, for the sake of simplicity we describe here a greedy approximation of the algorithm that comes close to the optimal in most cases.)

```
Find-coefficients at $G$
    Find-partitions:
        Initially each subset $H$ of $G$ defines its own partition.
        For each subset $H$ starting from least detailed group-by ALL.
            Find an immediate un-matched parent $P$ of $H$ resolving ties
                by choosing one sorted in the same order as $H$.
            If a valid parent found,
                merge $P$ to partition of $H$ and sort $P$ in same order as $H$.
    Perform-subtraction:
        For each partition $S$ of subsets
            sort $G$ in the order of group-bys in the partition.
            subtract all group-bys in $S$ from $G$ in one pass of $G$.
```

Example. Assume after the up-phase each of the group-bys are sorted in their lexicographic order. We partition the subsets for the down-phase of ABC as follows: First, ALL and A are merged to belong to same partition. Then, we merge ALL, A and AB in one partition, B and BC in second partition and C and AC in the third partition. All subsets are sorted in the right order except AC which is re-sorted in the order CA. During the subtraction phase, for the first partition ABC is already sorted in the right order, for the second we sort it in the order BCA and for the third we sort in the order CAB.

Analysis. For a n attribute group-by we cannot have less than $C(n, n/2) = \frac{n!}{(n/2)!(n/2)!}$ partitions of its subsets. This is so because the number of subset group-bys of size $n/2$ is $C(n, n/2)$ and none of these can be subsets of each other. Note that although $C(n, n/2)$ is much smaller than $2^n - 1$, it is still exponentially larger than n.

4.1.3. Rewriting. We now discuss further ways of speeding up computation by rewriting Eq. 3.3. Instead of the $2^n - 1$ terms in Eq. 3.3, we express the expected value as a sum of n terms as follows:

$$\hat{\ell}_{i_1 \dots i_n} = g^1 + \dots + g^n, \; where \; g^r = \text{avg}_{i_r}(\ell_{i_1 \dots i_n} - g^1 - \dots - g^{r-1}) \tag{4.1}$$

As an example, consider a cube with three dimensions A, B, C.

$$\hat{\ell}_{ijk} \quad = \quad g^1_{ij} + g^2_{ik} + g^3_{jk},$$

where,

$$\begin{aligned} g^1_{ij} &= \text{avg}_k(\ell_{ijk}) \\ g^2_{ik} &= \text{avg}_j(\ell_{ijk} - g^1_{ij}) \\ g^3_{jk} &= \text{avg}_i(\ell_{ijk} - g^1_{ij} - g^2_{ik}). \end{aligned}$$

The coefficients from the original Eq. 3.3 can be rewritten in terms of the new coefficients as:

$$\begin{aligned} r_{ijk} &= \ell_{ijk} - (g^1_{ij} + g^2_{ik} + g^3_{jk}) \\ \gamma_{ij} &= g^1_{ij} - g^1_i - g^1_j, \text{where} \\ & \quad g^1_i = \text{avg}_j(g^1_{ij}), \\ & \quad g^1_j = \text{avg}_i(g^1_{ij} - g^1_i), \\ \gamma_{ik} &= g^2_{ik} - g^2_k, \text{where} \\ & \quad g^2_k = \text{avg}_i(g^2_{ik}), \\ \gamma_{kj} &= g^3_{jk} \\ \gamma_i &= g^1_i - g^1, \text{where} \\ & \quad g^1 = \text{avg}_i(g^1_i) \\ \gamma_j &= g^1_j \\ \gamma_k &= g^2_k \\ \gamma &= g^1 \end{aligned}$$

Hierarchies can be incorporated in this equation easily. The equation would still contain k terms for a k dimensional cube. The only difference is that instead of averaging along *all* the values along a dimension, we average together elements that belong to the next level of the hierarchy along that dimension.

Lemma 4.1. *Equations 3.3 and 4.1 yield the same set of residuals when the cube contains no missing data.*

Proof. Replace the coefficients in Eq. 3.3 with the formula for their estimates in section 3.4. Thus, for three dimensions the equation takes the form:

$$\hat{\ell}_{ijk} = \ell_{+++} - \ell_{i++} - \ell_{+j+} - \ell_{++k} + \ell_{ij+} + \ell_{i+k} + \ell_{+jk} \tag{4.2}$$

This can be generalized to arbitrary number of dimensions where for each group-by we have a term with a positive coefficient when an even number of dimensions are aggregated (denoted by $+$) and a negative coefficient otherwise.

Now consider Eq. 4.1. We prove the equivalence of the rewrite step using induction. The rewrite holds for $n = 1$. Let us assume that it holds for some $n = k$. Thus, the RHS of Eq. 4.1 is equivalent to the RHS of Eq. 4.2. For $n = k + 1$, divide the terms in Eq. 4.2 into two parts: part 1 contains terms not aggregated along $(k+1)$th dimension and part 2 contains the remaining terms. The sum of the first k terms $g^1 \dots g^k$ of Eq. 4.1 gives us *all* the terms in part 1, i.e., those not aggregated along the $k+1$th dimension (by induction). The term g^{k+1} is equal to $\text{avg}_{i_{k+1}}(\ell_{i_1 \dots i_k i_{k+1}} - g^1 - \dots - g^k) = \text{avg}_{i_{k+1}}(\ell_{i_1 \dots i_k i_{k+1}}) - \text{avg}_{i_{k+1}}(g^1 + \dots + g^k)$ when the cube contains no missing values. The term $\text{avg}_{i_{k+1}}(g^1 + \dots + g^k)$ gives all terms of Eq. 4.2 aggregated along dimension $k+1$ and at least one other dimension. Thus, together $g_1 \dots g_k$ and g_{k+1} yield all the relevant terms. ■

When a cube does contain missing data, the residuals could differ depending on the number of missing values. One should evaluate the coefficients iteratively ([HMJ88], chapter 4) for producing accurate least squares fit in such cases. However, these methods are not practical for our goal of pre-mining an entire large database since they require multiple passes (often 10 or more) of data. Our implementation ignores the missing values in the calculation of the coefficients in both equations by calculating the average only over the values actually present.

Computing with Eq. 4.1. The rewritten formula can be computed as follows. First compute g^1 by averaging the starting $\ell_{i_1 \dots i_n}$ values along dimension i_n, subtract values g^1 from the original ℓ values, average the subtracted value along dimension i_{n-1} to compute g^2, subtract the values at g^2 from the modified ℓ values and so on until all dimensions are aggregated. The final ℓ value directly gives us the residual. Next, compute the other coefficients of the equation by recursively repeating the process for higher level aggregates on the average g values just calculated. These operations can be overlapped with the computation of the `user-agg` function of phase-1 as follows:

```
Compute($G$)
    Mark $G$ as computed.
    For each immediate child $H$ of $G$ not marked computed
        Compute and store the user-agg and avg-g values at $H$ from $G$
        Subtract the avg-g value at $H$ from $G$
    For each $H$ above
        Compute($H$) /* on the avg-g values. */
    Initial call: Compute(Base level cube)
```

The above procedure is first invoked on the base level group-by $A_1 A_2 \dots A_n$ and thereafter all the coefficients are obtained at their respective levels recursively. Note that each group-by G is computed only once in the above procedure.

Example. In Figure 7 we show the example of a three attribute group-by and the sequence of computations needed for getting its coefficients and residuals. An upward arrow denotes the averaging phase and a downward arrow denotes the subtraction phase. The numbers beside each edge denotes the order in which these operations are performed. We first average ABC along C to obtain AB, subtract the values at AB from ABC, average ABC along B to obtain AC, and so on until BC is subtracted from ABC. Next, we compute the coefficient at AB by averaging its g values along dimension B to obtain A, subtract out the results from AB and so on. When computing coefficient at AC we do not average and subtract along A because A has already been computed by averaging AB.

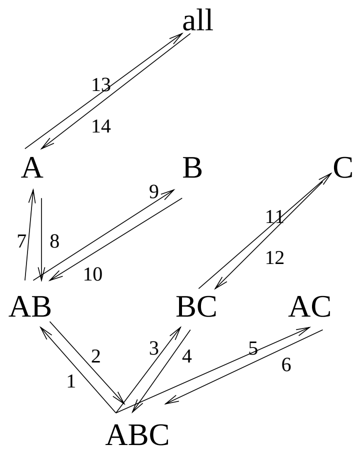


Figure 7: Fitting single equation for a three-attribute cube

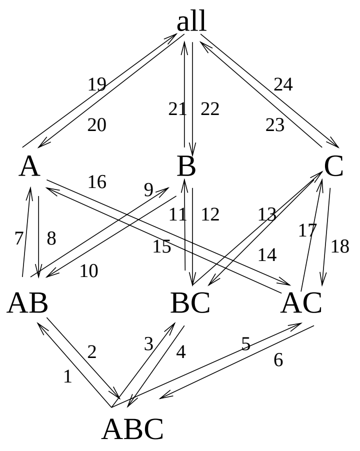


Figure 8: Fitting multiple equations for a three-attribute cube

Benefits of rewriting. The advantage of rewriting Eq. 3.3 into Eq. 4.1 as above is three fold. First we can compute the residuals of a k-dimensional cube by joining it with at most k other group-bys instead of $2^k - 1$ group-bys as in Eq. 3.3, an exponential difference in the number of join (subtraction) operations. Second, we can compute the residuals the same time as we compute the aggregate values in phase 1 and thus save on the sorting and comparison costs. Finally, there is no additional sorting cost beyond cube computation since, unlike in the UpDown method, the subtraction operation is followed immediately after the aggregation operation. For instance, when we sort ABC in the order BCA to compute BC, we can directly subtract the BC values from BCA and thus do not have to incur any additional sort cost on ABC for the subtraction phase. Thus, not only are the number of join operations exponentially smaller but also the cost of each join is significantly reduced since the joins require the same sorting order as the aggregation that precedes it.

Alternative Rewritings. There are other ways in which we could have rewritten Eq.3.3. For instance, for $n = 3$ another way of rewriting the equation is:

$$\begin{aligned} \hat{\ell}_{ijk} &= g_{ij}^{AB} + \gamma_k^C + \gamma_k^{AC} + \gamma_k^{BC}, \text{where} \\ g_{ij}^{AB} &= \text{avg}_k(\ell_{ijk}) \end{aligned}$$

The above equation uses four terms whereas Eq. 4.1 requires only three.

The goal in rewriting the equation in terms of as few coefficients as possible is to reduce the computation cost. Eq. 4.1 involves the fewest number of terms in each equation. It is because any equation equivalent to Eq. 3.3 must contain at least n terms since we must have at least one term from each of the $n-1$ dimensional group-bys.

4.2. Simultaneous computation of multiple equations

We can adapt our method for fitting single equations to the case where we fit simultaneously multiple equations — one for each group-by of the cube. A naive solution is to independently solve the single-equation problem for each group-by. But we can reduce cost by overlapping the model fitting step of multiple group-bys.

For the UpDown method, we can overlap as follows: For each k attribute group-by of the cube in the `up-phase`, compute and store the average ℓ value for each of the $2^{n-k}-1$ parents of the group-by instead of just the base level group-by. This implies that in the `up-phase` each group-by will have to compute averages along *all* of its immediate parents instead of just one of them. For instance for a 3-D cube, for group-by A we will need to store three coefficients for the equations at group-bys AB, AC and ABC and for group-by ALL we need to store seven coefficients. This is inefficient both computationally and storage-wise.

The rewrite method, enables us to fit multiple equations much more efficiently. We proceed bottom up and first compute the residuals for the bottom-most group-by using the aggregation and subtraction operations with respect to its immediate children group-bys as in the single equation case. At the end of this, the residuals for the bottom-most group-by are already computed. Thus, we can drop the g terms calculated so far and start to fit the equations of the $n-1$ attribute group-bys on the aggregated function `user-agg`. Each of these $n-1$ dimensional group-bys can now be treated independently and we can recursively fit equations for each group-by of the cube as shown in the pseudo-code below.

```
ComputeMulti($G$)
    For each immediate child $H$ of $G$
        Compute the avg-g values at $H$ by aggregating $G$
        If $G$ is the smallest parent of $H$
            also, compute and store user-agg function along with above step
        Subtract the avg-g value at $H$ from $G$
    For each $H$ whose user-agg function computed above
        ComputeMulti($H$) /* on the user-agg function values. */
Initial call: ComputeMulti(Base level cube)
```

Note the two key differences between the routine Compute() for the single equation case and ComputeMulti() for the multi equation case. First, for each group-by *all* of its immediate children are used instead of just the un-computed ones as in the single equation case. Second, for each group-by we start from the aggregate function value for that group-by rather than the g values computed from the previous group-by.

Example. Figure 8 shows the sequence of aggregation and subtraction operations that happen when fitting multiple equations using the rewrite procedure. The numbers beside each edge denotes a correct order in which these operations could be performed. The residuals at ABC are computed after step 6. Step 7 then operates on the aggregate values at AB and so on.

4.3. Estimating variance

The estimation of variance requires us to solve Eq. 3.4. Since a closed form solution of this equation is not available, we solve the equation iteratively. To avoid making multiple passes of the data, we evaluate the equation for a fixed set of values of ρ (10 in our case equally spaced between 0 and 3). The final value is the point in between the two points where the sign changes. If the value of ρ happens to fall outside the chosen range a re-scan of the data with the expanded range is necessary. For none of our experiments so far we had to rescan. For calculating the variance we do not have to incur additional data scans because in both the rewrite and the UpDown method we know when the last subtraction for a term is completed and thus can use the residuals to directly perform the variance calculations.

4.4. Experiments

We study the performance of model-fitting and show how the gap between the modified UpDown and the rewrite schemes increases as we increase the number of dimensions. We also compare the total time to find exceptions with the time to compute aggregates in a data cube.

Experiments with synthetic datasets. We first report our experiments with synthetic datasets used to study the trend as the number of dimensions is scaled up.

Each dataset consists of one million tuples and all dimensions of the cube have equal number of members. Of the total number of dimension combinations possible 20% are assumed to be present. These points are assumed to be distributed uniformly randomly in the cube. The exact value of the measure is not important since we are only interested in the running time. The number of dimensions is varied from 3 to 6.

Figure 10 shows the time to do model-fitting using the rewrite scheme relative to the modified UpDown scheme (MUD) for the single equation case. We also show the time to simply compute all the group-bys (Agg). The total execution time is normalized by the time taken by "Agg". The Figure shows that as we increase the number of dimensions the difference between the rewrite and the UpDown methods of model-fitting becomes even wider. However, the gap between model-fitting using UpDown scheme and aggregate computation does not increase much and remains within 20% in all cases. This confirms our earlier conclusions based on the complexity analysis.

Real-life datasets. In these experiments we study the total time taken to find exceptions and compare it with the aggregate computation time. Exception finding time includes all the three phases of aggregate computation, model-fitting and exception summarization.

Datasets. The three datasets we used in our experiments were derived from sales transactions of various department stores and mail order companies. A brief description is given

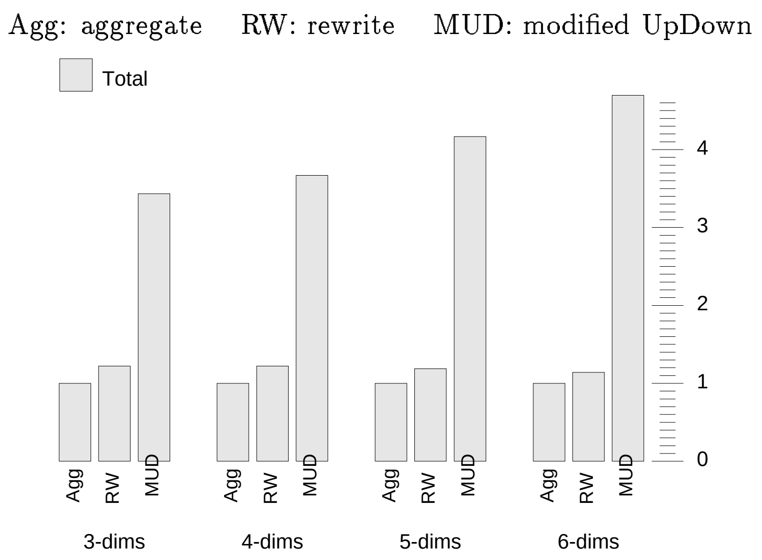


Figure 9: Time to do model-fitting using the rewrite and UpDown methods compared against simple aggregate computation. The y-axis denotes the total time normalized by the time taken by "Agg" for each dataset.

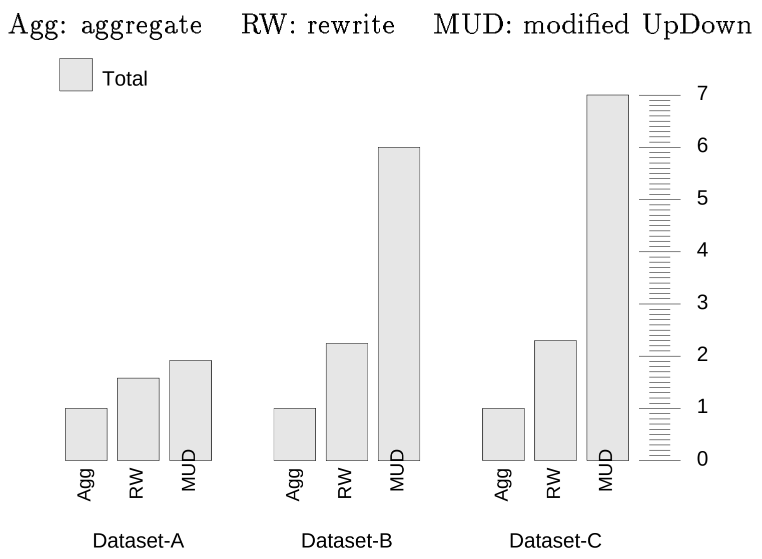


Figure 10: Time to run various algorithms on three real life datasets. The y-axis denotes the total time normalized by the time taken by "Agg" for each dataset. RW and MUD also include the time to summarize exceptions.

next. The datasets differ in the number of transactions, the number of attributes, and the number of distinct values for each attribute.

- **Dataset-A:** This data is about supermarket purchases. Each transaction has three attributes: store, date and product category. There are a total of 5 million transactions.
- **Dataset-B:** This data is from a mail order company. A sales transaction here consists of three attributes: the customer identifier, the order date (month and year) and the product. In addition, there is hierarchy on product dimension that groups products based on their category and a hierarchy on the date dimension that groups the monthly dates to "years". There are a total of one million transactions.
- **Dataset-C:** This is data about grocery purchases of customers from a supermarket. Each transaction has four attributes: the date of purchase, the shopper type, the store code and the product group of the item purchased. There are a total of one million transactions.

Figure 10 shows the total time to find exceptions using the rewrite(RW) and modified UpDown(MUD) methods compared with aggregate computation (Agg). From this figure we can make the following observations:

- The performance improves almost three-fold by using the rewrite technique when compared with the UpDown scheme.
- The improvement is higher for dataset-B and C than dataset-A since dataset-A consists of only three dimensions and no hierarchies. For dataset-A, the number of passes needed on the base cube (the largest group-by in the cube) for the `down-phase` of the UpDown method is $C(k, k/2) = 3$ when $k = 3$. This is same as the number of passes on the base cube for the rewrite scheme. There is still a small improvement because for the rewrite scheme we need no additional sorting over "Agg" computation whereas for the UpDown scheme at least two additional sorts are needed.
- After the rewriting, the time to find exceptions and summarize them comes to within a factor of 2.2 of the time to compute group-bys. Thus, computationally, pre-mining for exceptions in OLAP data is within tolerable expense.

Performance when fitting multiple equations. We also measured the time to fit multiple equations at all group-bys of the cube. Drawing upon the performance results for the single equation case, we only experimented with the rewrite method. We found that for the rewrite scheme time to fit multiple equation comes to within 10% of the time to fit single equations for all the real-life and synthetic datasets. This is expected because the extra work needed for fitting multiple equations arises only at upper levels of aggregations where the size of the group-bys is much smaller than the lower level group-bys (compare figures 7 and 8 as an example).

Prod.Grp	Tool
Prod.Cat	PDT
Prod.Name	DBMS Engines (Non-Object)
Plat.Users	M
Plat.Type	UniM
Plat.Name	Multiuser UNIX SunSoft Solaris

Average of Revenue	Years				
Geography	1990	1991	1992	1993	1994
Asia/Pacific	2.42	2.92	5.20	8.00	19.84
Rest of World	2.64	5.23	10.02	15.12	23.90
United States	19.61	27.18	40.40	62.31	5.30
Western Europe	14.32	23.78	36.80	55.95	78.29

Figure 11: Exceptions in the S^3GP^3Y plane of the software dataset.

4.5. Experience with software revenue data

An important step in the evaluation of a mining technique is not simply to measure its performance but also to check whether it found anything interesting or surprising in real-life datasets. In Section 2 we showed some qualitative results with a dataset that the OLAP vendor Essbase ships as an example of a typical OLAP dataset. In this section we report on some interesting exceptions we found with another real-life dataset. This dataset (available from the International Data Corporation and also the Cognos web site [Cor97]) is about the total yearly revenue of various software products on different platforms across different regions of the world for years 1990 to 1994. The dimensions are as follows. The numbers within brackets give the number of members for each dimension.

- Software products, S: 3 level hierarchy [Category(3), $S^1 \rightarrow$ Group(14), $S^2 \rightarrow$ Name(68), S^3]
- Geography, G: no hierarchy, 4 members
- Platform, P: 3 level hierarchy [Number of users (2), $P^1 \rightarrow$ type(16), $P^2 \rightarrow$ name(43), P^3]
- Year, Y: no hierarchy, 5 members (1990 to 1994).

The data has 18115 points out of 58480 possible. The function used for aggregating data at higher levels is average. We present two of the interesting exceptions found using our methodology:

In Figure 11 we show the revenue figures for DBMS Engines on the Multiuser UNIX Sunsoft Solaris platform. Notice that, in 1994, in US, the revenue of DBMS products dropped by 1/10th on the Sun Solaris platform. We mark this as an exception because it stands out along *all* different views of the data. On comparing this exception along each of the four dimensions while keeping the other three fixed we find:

1. In other geographies, revenue increased by at least 30% for DBMSs on Sun Solaris.

Plat.type	(All)
Prod.name	(All)
Plat.name	(All)
Plat.users	(All)
Year	(All)
Prod.Grp	SySW

Average of Sales	Geography			
Prod.Cat	Asia/Pacific	Rest of World	United States	Western Europe
DCS	2.78	3.87	9.36	7.87
Mdw	0.99	0.09	0.60	0.30
Oth	0.38	0.25	1.70	0.74
SMS	2.95	2.86	8.31	6.26
SSW	20.64	23.10	42.11	36.39
SUt	4.35	4.48	10.79	9.84
SyI	7.19	0.28	1.82	0.97
UMO	9.78	10.81	30.35	21.04

Figure 12: Exceptions in the S^2G plane of the software dataset.

2. For other products in the same category, revenue increased by at least 20% in US on Sun Solaris.

3. On most other platforms, revenue increased by at least 30% for DBMSs in US.

4. For other years, US accounts for almost 40% revenue of DBMSs on Sun Solaris whereas it is only second largest in 1994.

Note that this exception occurs at the lowest level of detail and it might have been almost impossible for a manual inspection process to find this big drop since at the upper levels of aggregations the impact of this drop is not significant.

There are exceptions at higher levels of aggregations too. In Figure 12 we show an exception found in the group-by (Product.category, Geography) for the category `SyI` (System Integration software) and the Asia/pacific region. This is an exception since overall, Asia/Pacific accounts for only 12% of the revenue whereas it is 70% just for this product category. On drilling down, we observe that this trends holds for all platforms and instead of marking each of these points as an exception, we report a single exception at a higher aggregation level.

5. Conclusion

5.1. Contribution

- We developed a novel method of effectively navigating large OLAP data cubes. Our method guides the user to interesting regions exhibiting anomalous behavior using pre-computed exceptions. This method enhances a user's capability of discovering interesting areas in the data compared with the current manual discovery, and greatly reduces the number of drill-down and selection steps that analysts currently go through – especially in large cubes with many different dimensions and hierarchies. We are not aware of any OLAP system that currently offers this capability.

- We presented the statistical foundation of our methodology for identifying exceptions, which was chosen after considering a number of other competing techniques and suitably adapted so as to best match the requirements of OLAP datasets. This methodology seamlessly integrates hierarchies and finds exceptions at multiple levels of aggregations. The coefficients at different levels of the cube have the property that they reflect adjustments to the combined average value obtained from higher level aggregations of the cube. As the user typically navigates the data cube top-down, this enables the user to very naturally capture the context in which the value was declared an exception.

- We devised methods of efficiently computing exceptions. Novel rewriting techniques are used to reduce the cost of model fitting and modifying the computation flow so as to mesh exception finding with cube computation. These optimizations enable us to fit separate equations at each of the different group-bys of the cube in almost the same time it takes to compute aggregates — this is something which was not evident when we started work on the problem.

- Our notion of exceptions ties in well with the cube metaphor and thus enables easy integration into existing MOLAP and ROLAP products. The process of finding exceptions involves the same kind of aggregations and scan operations as normal aggregate precomputation. The exception themselves can be stored, indexed and retrieved just like precomputed aggregates. Our prototype on Essbase has been implemented using this same principle.

- We showed how using our techniques we found interesting exceptions in two OLAP datasets. We have also applied the proposed method to two rather unconventional OLAP datasets. The first data-set was obtained from an auto insurance company and we used our prototype to study exceptional patterns relating age, marital status, profession and location to the number of insurance claims a person makes. The second data set was obtained from a chain store that sells both catalog and retail goods. We found exceptions at different levels to identify regions of exceptionally high or low market shares of retail and catalog sales amongst different income levels, age groups and geographical locations. These studies further validated the utility of the proposed approach.

5.2. Future work

- Special treatment of time dimension: Time is an ordered dimension and it is possible to apply special time series analysis techniques [Cha84] to further refine the notion of exceptions. We are working on ways to enhance our model equation to integrate these order specific terms.

- Model selection: It is possible that the full model equation with all its terms may lead to over-fitting and a simpler model could provide a better fit. We are experimenting with two schemes for model selection: one based on the minimum description length principle [Ris87] and the other based on statistical tests of goodness of fit [BFH75].

- User customization: In some situations users may want to influence the automated process with their own domain-dependent notion of exceptions, for instance, providing different weights to variations along different dimensions. The challenge in providing support for user customization is finding an appropriate expression language that can integrate with our statistical notion. Also, users might want to find exceptions only in certain regions of the cube and not in others. This customization is relatively easy to incorporate.

References.

[AAD$^+$96] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. of the 22nd Int'l Conference on Very Large Databases*, pages 506–521, Mumbai (Bombay), India, September 1996.

[AAR96] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. A linear method for deviation detection in large databases. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[AGS97] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling multidimensional databases. In *Proc. of the 13th Int'l Conference on Data Engineering*, Birmingham, U.K., April 1997.

[Arb] Arbor Software Corporation. *Application Manager User's Guide, Essbase version 4.0.* `http://www.arborsoft.com`.

[BFH75] Y. Bishop, S. Fienberg, and P. Holland. *Discrete Multivariate Analysis theory and practice.* The MIT Press, 1975.

[Cha84] C. Chatfield. *The analysis of time series.* Chapman and Hall, 1984.

[CL86] William W. Cooley and Paul R Lohnes. *Multivariate data analysis.* Robert E. Krieger publishers, 1986.

[Col95] George Colliat. OLAP, relational, and multidimensional database systems. Technical report, Arbor Software Corporation, Sunnyvale, CA, 1995.

[Cor97] Cognos Software Corporation. Power play 5, special edition. tthttp://www.cognos.com/powercubes/index.html, 1997.

[GBLP96] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tabs and sub-totals. In *Proc. of the 12th Int'l Conference on Data Engineering*, pages 152–159, 1996.

[HMJ88] D. Hoaglin, F. Mosteller, and Tukey. J. *Exploring data tables, trends and shapes.* Wiley series in probability, 1988.

[HMT83] D.C. Hoaglin, F. Mosteller, and J.W. Tukey. *Understanding Robust and Exploratory Data Analysis.* John Wiley, New York, 1983.

[JD88] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[Joh92] R.A. Johnson. *Applied Multivariate Statistical Analysis.* Prentice Hall, 1992.

[Klo95] W. Klosgen. Efficient discovery of interesting statements in databases. *Journal of Intelligent Information Systems: Integrating Artificial Intelligence and Database Technologies*, 4(1):53–69, 1995.

[Man71] J. Mandel. A new analysis of variance model for non-additive data. *Technometrics*, 13:1–18, 1971.

[Mon91] D.G. Montgomery. *Design and Analysis of Experiments*, chapter 13. John Wiley & sons, third edition, 1991.

[OLA96] The OLAP Council. *MD-API the OLAP Application Program Interface Version 0.5 Specification*, September 1996.

[Ris87] Jorma Rissanen. Complexity and information in contingency tables. In *Proc. Second International tampere conference in statistics*, pages 289–302, Tampere, Finland, June 1987.

[RL87] Peter J. Rousseeuw and Annick M. Leory. *Robust regression and outlier detection.* Wiley series in probability and statistics, 1987.

[SD90] Jude W. Shavlik and Thomas G. Dietterich. *Readings in Machine LearningSeries in Machine Learning.* Morgan Kaufmann, 1990.