Discovery-driven Exploration of OLAP Data Cubes*

Sunita Sarawagi Rakesh Agrawal Nimrod Megiddo

IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA

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.

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”

* This is an abridged version of the full paper that appears in [SAM98].

(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.

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.

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. Refer to [SAM98] for some performance results and experience with real-life datasets that illustrates the effectiveness of the proposed approach. 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 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.

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%

Figure1. Change in sales over time

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

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%

Figure2. Change in sales over time for each product

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 different Regions. By highlighting

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%

Figure3. Change in sales of Product “Diet-Soda” over time in each Region

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.

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%

Figure4. Change in sales over Time for Product “Cola” in different Region

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”.

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. In [SAM98] we discuss the alternatives we considered before deciding on our approach.

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%

Figure5. Change in sales over Time for Product "Birch-B"

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, ⟨Birch-B,Oct⟩ is an exception at the ⟨Product,Month⟩ group-by (Figure 2) and that means we do not need to mark as exceptions all the high values for ⟨Birch-B,Oct, *⟩ at the ⟨Product,Month,State⟩ group-by (Figure 5).
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. 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:

$$\hat{y}_{i_1 i_2 \ldots i_n} = f(\gamma^G_{(i_r | d_r \in G)} | G \subset \{d_1, d_2, \ldots d_n\}) \tag{1}$$

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. 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

$$s_{i_1 i_2 \ldots i_n} = \frac{|y_{i_1 i_2 \ldots i_n} - \hat{y}_{i_1 i_2 \ldots i_n}|}{\sigma_{i_1 i_2 \ldots i_n}} \tag{2}$$

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. 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. 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. (See [SAM98] for an intuitive reason for this.) 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

$$\hat{l}_{i_1 i_2 \ldots i_n} = \log \hat{y}_{i_1 i_2 \ldots i_n} = \sum_{G \subset \{d_1, d_2, \ldots d_n\}} \gamma^{G}_{(i_r | d_r \in G)} \tag{3}$$

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

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

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.
   - $\gamma^{A_r}_{i_r} = \ell_{+..+i_r+..+} - \gamma$ where $\ell_{+..+i_r+..+}$ is the mean over all values along i_rth member of dimension A_r. Thus, $\gamma^{A_r}_{i_r}$ denotes how much the average of the values along i_rth member of dimension A_r differs from the overall average.
   - $(\gamma)^{A_r A_s}_{i_r i_s} = \ell_{+..+i_r+..+i_s+..+} - \gamma^{A_r}_{i_r} - \gamma^{A_s}_{i_s} - \gamma$.

   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:

$$\sum \frac{(y_{i_1 i_2 \ldots i_n} - \hat{y}_{i_1 i_2 \ldots i_n})^2}{(\hat{y}_{i_1 i_2 \ldots i_n})^\rho} \cdot \log \hat{y}_{i_1 i_2 \ldots i_n} - \sum \log \hat{y}_{i_1 i_2 \ldots i_n} = 0 \ . \tag{4}$$

The method we used for solving the equation to find ρ is discussed in [SAM98].

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. We present concise definitions of the $\mathcal{S}$elfExp,$\mathcal{I}$nExp and $\mathcal{P}$athExp quantities we associate with each cell for this purpose. In [SAM98] more formal definitions appear.

$\mathcal{S}$elfExp: denotes the exception value of the cell. This quantity is defined as the scaled absolute value of the residual defined in Eq. 2 with a cut-off threshold of τ.

$\mathcal{I}$**nExp:** denotes the total degree of surprise over *all* elements reachable by drill-downs from this cell. We define it formally as the maximum $\mathcal{S}$elfExp value over all cells underneath this cell.

$\mathcal{P}$**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.

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 with a few differences as discussed in [SAM98].

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 and later present improvements.

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 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 `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. This straightforward computation can be improved further as discussed in [SAM98].

Rewriting We now discuss further ways of speeding up computation by rewriting Eq. 3. Instead of the $2^n - 1$ terms in Eq. 3, we express the expected value as a sum of n terms as follows:

$$\hat{\ell}_{i_1 \dots i_n} = g^1 + \dots + g^n, \ where \ g^r = \text{avg}_{i_r}(\ell_{i_1 \dots i_n} - g^1 - \dots - g^{r-1}) \tag{5}$$

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

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

The coefficients from the original Eq. 3 can be rewritten in terms of the new coefficients as:

$$r_{ijk} = \ell_{ijk} - (g^1_{ij} + g^2_{ik} + g^3_{jk})$$

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

Lemma 1. *Equations 3 and 5 yield the same set of residuals when the cube contains no missing data. [Proof appears in [SAM98].]*

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. 5 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)
```

Example In Figure 6 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.

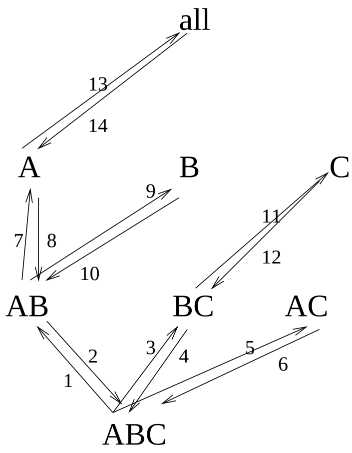

Figure6. Fitting single equation for a three-attribute cube

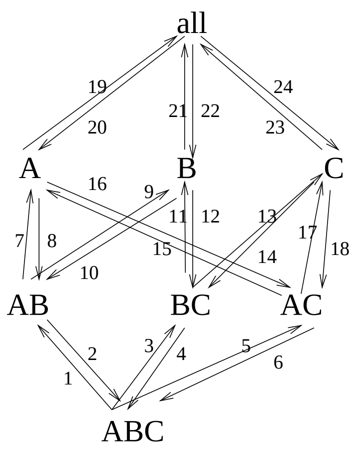

Figure7. Fitting multiple equations for a three-attribute cube

Benefits of rewriting The advantage of rewriting Eq. 3 into Eq. 5 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, 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. 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. For instance, for $n = 3$ another way of rewriting the equation is:

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

The above equation uses four terms whereas Eq. 5 requires only three.

The goal in rewriting the equation in terms of as few coefficients as possible is to reduce the computation cost. Eq. 5 involves the fewest number of terms in each equation. It is because any equation equivalent to Eq. 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. 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 7 shows the sequence of aggregation and subtraction operations that happen when fitting multiple equations using the rewrite procedure.

5 Conclusion

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.

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. 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. In [SAM98] we present how our model handles hierarchies and ordered dimensions like time.

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. Our experiments (detailed in [SAM98]) show that these techniques yield almost a factor of three to four performance improvement. We have applied our technique on several real-life OLAP datasets with interesting results. In [SAM98] we report some of these findings.

Future work in the area should incorporate methods for model selection and user customization of the definition of exceptions.

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.

[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.

[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.

[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.

[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.

[SAM98] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven exploration of OLAP data cubes. Research Report RJ 10102 (91918), IBM Almaden Research Center, San Jose, CA 95120, January 1998. Available from `http://www.almaden.ibm.com/cs/quest`.

This article was processed using the LaTeX macro package with LLNCS style