Explaining differences in multidimensional aggregates

Sunita Sarawagi*

School of Information Technology
Indian Institute of Technology, Bombay
Mumbai 400076, INDIA
`sunita@cs.berkeley.edu`

Abstract

Our goal is to enhance multidimensional database systems with advanced mining primitives. Current Online Analytical Processing (OLAP) products provide a minimal set of basic aggregate operators like sum and average and a set of basic navigational operators like drill-downs and roll-ups. These operators have to be driven entirely by the analyst's intuition. Such ad hoc exploration gets tedious and error-prone as data dimensionality and size increases. In earlier work we presented one such advanced primitive where we premined OLAP data for exceptions, summarized the exceptions at appropriate levels, and used them to lead the analyst to the interesting regions.

In this paper we present a second enhancement: a single operator that lets the analyst get summarized reasons for drops or increases observed at an aggregated level. This eliminates the need to manually drill-down for such reasons. We develop an information theoretic formulation for expressing the reasons that is compact and easy to interpret. We design a dynamic programming algorithm that requires only one pass of the data improving significantly over our initial greedy algorithm that required multiple passes. In addition, the algorithm uses a small amount of memory independent of the data size. This allows easy integration with existing OLAP products. We illustrate with our prototype on the DB2/UDB ROLAP product with the Excel Pivot-table frontend. Experiments on this prototype using the OLAP data benchmark demonstrate (1) scalability of our algorithm as the size and dimensionality of the cube increases and (2) feasibility of getting interactive answers even with modest hardware resources.

*Part of the work was done when the author was at IBM Almaden Research Center, USA

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

1 Introduction

Online Analytical Processing (OLAP) [Cod93, CD97] products [Arb, Sof] were developed to help analysts do decision support on historic transactional data. Logically, they expose a multidimensional view of the data with categorical attributes like Products and Stores forming the *dimensions* and numeric attributes like Sales and Revenue forming the *measures* or cells of the multidimensional cube. 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. The measure attributes are aggregated to various levels of detail of the combination of dimension attributes using functions like sum, average, max, min, count and variance. For exploring the multidimensional data cube there are navigational operators like select, drill-down, roll-up and pivot conforming to the multidimensional view of data. In the past most of the effort has been spent on expediting these simple operations so that the user can interactively invoke sequences of these operations. A typical such analysis sequence proceeds as follows: the user starts from an aggregated level, inspects the entries visually maybe aided by some graphical tools, selects subsets to inspect further based on some intuitive hypothesis or needs, drills down to more detail, inspects the entries again and either rolls up to some less detailed view or drills down further and so on.

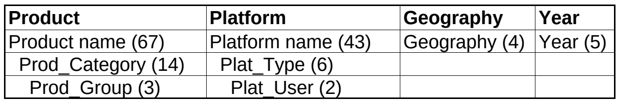
The above form of manual exploration can get tedious and error-prone for large datasets that commonly appear in real-life. For instance, a typical OLAP dataset has five to seven dimensions, an average of three levels hierarchy on each dimension and aggregates more than a million rows [Cou]. The analyst could potentially overlook significant discoveries due to the heavy reliance on intuition and visual inspection.

This paper is part of our continued work on taking OLAP to the next stage of interactive analysis where we automate much of the manual effort spent in analysis. Recent work in this direction attempt to enhance OLAP products with known mining primitives: decision tree classifiers to find the factors affecting profitability of products is used by Information discovery Inc [Dis] and Cognos [Cor97a], clustering customers based on buying patterns to create new hierarchies is used in Pilot Software [Sof]; and association rules at multiple levels of aggregation to find progressively detailed correlation between members of a dimension is suggested by Han et al [HF95]. In all these cases, the approach has been to take existing mining algorithms and integrate them within OLAP products. Our approach is different in that we first investigate how and why analysts currently explore the data cube and next automate them using new or previously known operators. Unlike the batch processing of existing mining algorithms we wish to enable interactive invocation so that an analyst can use them seamlessly with the existing simple operations.

In [SAM98] we presented one such operation that was motivated with the observation that a significant reason why analysts explore to detailed levels is to search for abnormalities or exceptions in detailed data. We proposed methods for finding exceptions and used them to guide the analysts to the interesting regions.

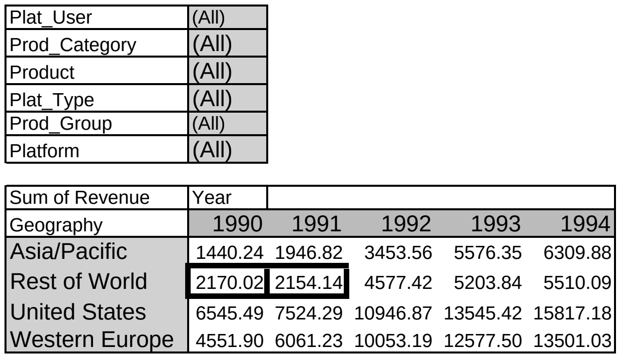
In this paper we seek to automate another area where analysts spend significant manual effort exploring the data: namely, exploring reasons for why a certain aggregated quantity is lower or higher in one cell compared to another. For example, a busy executive looking at the annual reports might quickly wish to find the main reasons why sales dropped from the third to the fourth quarter in a region. Instead of digging through heaps of data manually, he could invoke our new DIFF operator which in a single step will do all the digging for him and return the main reasons in a compact form that he can easily assimilate.

We explore techniques for *defining* how to answer these form of "why" queries, *algorithms* for efficiently answering them in an ad hoc setting and system issue in integrating such a capability with existing OLAP products.



Product	Platform	Geography	Year
Product name (67)	Platform name (43)	Geography (4)	Year (5)
Prod_Category (14)	Plat_Type (6)		
Prod_Group (3)	Plat_User (2)		

Figure 1: Dimensions and hierarchies of the software revenue data. The number in brackets indicate the size of that level of the dimension.



Plat_User	(All)
Prod_Category	(All)
Product	(All)
Plat_Type	(All)
Prod_Group	(All)
Platform	(All)

Sum of Revenue	Year				
Geography	1990	1991	1992	1993	1994
Asia/Pacific	1440.24	1946.82	3453.56	5576.35	6309.88
Rest of World	2170.02	2154.14	4577.42	5203.84	5510.09
United States	6545.49	7524.29	10946.87	13545.42	15817.18
Western Europe	4551.90	6061.23	10053.19	12577.50	13501.03

Figure 2: Total revenue by geography and year. The two boxes with dark boundaries ('Rest of World', year 1990 and 1991) indicate the two values being compared.

1.1 Contents

We first demonstrate the use of the new DIFF operator using a real-life dataset. Next in Section 2 we discuss ways of formulating the answers. In Section 3 we present algorithms for efficiently finding the best answer based on our formulation. In Section 4 we describe our overall system architecture and present experimental results.

1.2 Illustration

Consider the dataset shown in Figure 1 with four dimensions Product, Platform, Geography and Time and a three level hierarchy on the Product and Platform dimension. This is real-life dataset obtained from International Data Corporation (IDC) [Cor97b] about the total yearly revenue in millions of dollars for different software products from 1990 to 1994. We will use this dataset as a running example in the paper.

Suppose a user is exploring the cube at the Geography$\times$Year plane as shown in Figure 2. He notices a steady increase in revenue in all cases except when going from 1990 to 1991 in 'Rest of the World'.

With the existing tools the user has to find the reason for this drop by manually drilling down to the numerous different planes underneath it, inspecting the entries for big drops and drilling down further. This process can get rather tedious especially for the typically larger real-life datasets. We propose to use the new operator for finding the answer to this question in one step. The user simply highlights the two cells and invokes our "DIFF" module. The result as shown in Figure 3 is a list of at most 10 rows (the number 10

PRODUCT	PLAT_U	PLAT_T	PLATFORM	YEAR_1990	YEAR_1991	RATIO	ERROR
(All)-	(All)-	(All)	(All)	1620.02	1820.05	1.12	34.07
Operating Systems	Multi	(All)-	(All)	253.52	197.86	0.78	23.35
Operating Systems	Multi	Other M.	Multiuser Mainframe IBM	97.76	1.54	0.02	0.00
Operating Systems	Single	Wn16	(All)	94.26	10.73	0.11	0.00
*Middleware & Oth.(	Multi	Other M.	Multiuser Mainframe IBM	101.45	9.55	0.09	0.00
EDA	Multi	Unix M.	(All)	0.36	76.44	211.74	0.00
EDA	Single	Unix S.	(All)	0.06	13.49	210.78	0.00
EDA	Single	Wn16	(All)	1.80	10.89	6.04	0.00

Figure 3: Reasons for the drop in revenue marked in Figure 2.

Geography	(All)				
Plat_User	(All)				
Prod_Group	Soln				
Sum of Revenue	Year				
Prod_Category	1990	1991	1992	1993	1994
Cross Ind. Apps	1974.57	2484.20	4563.57	7407.35	8149.86
Home software				293.91	574.89
Other Apps	843.31	1172.44	3436.45		
Vertical Apps	898.06	1460.83	2826.90	7947.05	8663.39

Figure 4: Total revenue for different categories in product group "Soln". We are comparing the revenues in year 1992 and 1993 for product category "Vertical Apps".

is configurable by user) that concisely explain the reasons for the drop. In the figure the first row shows that after discounting the rows explicitly mentioned below it as indicated by the "-" symbol after (All), the overall revenue actually increased by 10%. The next four rows (rows second through fifth) identify entries that were largely responsible for the large drop. For instance, for "Operating systems" on "Multiuser Mainframe IBM" the revenue decreased from 97 to 1.5, a factor of 60 reduction and for "Operating systems" and all platforms of type "Wn16" the total revenue dropped from 94.3 to 10.7. The last three rows show cases where the revenue showed significantly more than the 10% overall increase indicated by the first row. The role of the RATIO and ERROR columns in Figure 3 will be explained in Section 2.

Similarly, a user might be interested in exploring reasons for sudden increases. In Figure 4 we show the total revenue for different categories in the Product group 'Soln'. The user wants to understand the reason for the sudden increase in revenue of "Vertical Apps" from 2826 in 1992 to 7947 in 1993 — almost a factor of three increase. For the answer he can again invoke the DIFF operator instead of doing multiple drill downs and selection steps on the huge amount of detailed data. The result is the set of rows shown in Figure 5. From the first row of the answer, we understand that overall the revenue increased by a modest 30% after discounting the rows underneath it. The next set of rows indicate that the main increase is due to product categories Manufacturing-Process, Other vertical Apps, Manufacturing-Discrete and Health care in various geographies and platforms. The last two rows indicate the two cases where there was actually a drop that is significant compared to the 30% overall increase indicated by the first row.

These form of answers helps the user to quickly grasp the major findings in detail data by simply glancing at the few rows returned.

2 Problem Formulation

In this section we discuss different formulations for summarizing the answer for the observed difference.

The user poses the question by pointing at two aggregated quantities v_a and v_b in the data cube and wishes to know why one is greater or smaller than the other. Both v_a and v_b are formed by aggregating one or more dimensions. Let C_a and C_b denote the two subcubes formed by expanding the common aggregated dimensions of v_a and v_b. For example, in Figure 2 v_a denotes the revenue for cell (All platforms, All products, 'Rest of world', 1990) and v_b denotes the total revenue for cell (All platforms, All products, 'Rest of world', 1991), C_a denotes the two-dimensional subcube with dimension $<$ Platform, Product $>$ for Year=1990 and Geography='Rest of world' and C_b denotes the two-dimensional subcube with the same set of dimensions for Year=1991 and Geography='Rest of world'. In this example, the two cells differ in only one dimension. We can equally well handle cases where the two cells differ in more than one dimension as long as we have some common dimensions on which both the cells are aggregated. The answer $\mathcal{A}$ consists of a set of rows that best explains the observed increase or decrease at the aggregated level. There is a limit N (configurable by the user) on the size of $\mathcal{A}$. The user sets this limit based on the number of rows the user is willing to inspect. We expect the value of N to be small around a dozen or so.

A simple approach is to show the large changes in detailed data sorted by the magnitude of the change (detail-N approach). The obvious problem with this simple solution is that the user could still be left with a large number of entries to eye-ball. In Figure 6 we show the first 10 rows obtained by this method for the difference query in Figure 4. This detail-N approach explains only 900 out of the total difference of about 5000. In contrast, with the answer in Figure 5 we could

PRODUCT	GEOGRAPHY	PLAT_TYPE	PLATFORM	YEAR_1992	YEAR_1993	RATIO	ERROR
(All)-	(All)-	(All)-	(All)	2113.0	2763.5	1.3	200.0
Manufacturing - Process	(All)	(All)	(All)	25.9	702.5	27.1	250.0
Other Vertical Apps	(All)-	(All)-	(All)	20.3	1858.4	91.4	251.0
Other Vertical Apps	United States	Unix S.	(All)	8.1	77.5	9.6	0.0
Other Vertical Apps	Western Europ	Unix S.	(All)	7.3	96.3	13.2	0.0
Manufacturing - Discrete	(All)	(All)	(All)		1135.2		0.0
Health Care	(All)-	(All)-	(All)-	6.9	820.4	118.2	98.0
Health Care	United States	Other M.	Multiuser Mai	1.5	10.6	6.9	0.0
Banking/Finance	United States	Other M.	(All)	341.3	239.3	0.7	60.0
Mechanical CAD	United States	(All)	(All)	327.8	243.4	0.7	34.0

Figure 5: Reasons for the increase in revenue marked in Figure 4.

PRODUCT	GEOGRAPHY	PLATFORM	YEAR_1992	YEAR_1993	RATIO
Other Vertical Apps	Western Europe	Multiuser Minicomputer OpenVMS		99.9	
Other Vertical Apps	Asia/Pacific	Single-user MAC OS		92.5	
Other Vertical Apps	Rest of World	Multiuser Mainframe IBM		88.1	
Other Vertical Apps	Western Europe	Single-user UNIX	7.3	96.3	13.2
Other Vertical Apps	United States	Multiuser Minicomputer Other		97.2	
Other Vertical Apps	United States	Multiuser Minicomputer OS/400		99.5	
Other Vertical Apps	Asia/Pacific	Multiuser Minicomputer OS/400		99.6	
EDA	Western Europe	Multiuser UNIX	192.6	277.8	1.4
Manufacturing - Discrete	United States	Multiuser Mainframe IBM		88.4	
Health Care	United States	Multiuser Minicomputer Other		88.2	

Figure 6: The top ten rows of the detail-N approach.

Product	Manufacturing - Process	
Geography	(All)	
Sum of Revenue	Year	
Plat_Type	1992	1993
Other M.	9.86	472.84
Other S.	0.02	21.88
Unix M.	7.45	105.09
Unix S.	1.31	16.89
Wn16	3.27	85.38
Wn32		0.38

Figure 7: Summarizing rows with similar change.

explain more than 4500 of the total difference. The main idea behind this more compact representation is summarizing rows with similar changes. In Figure 5 we have only one row from the detailed level the rest are all from summarized levels. By aggregating over Platforms and Geography with similar change, a more compact representation of the change is obtained. For instance, in Figure 7 we show the details along different "Platforms" for the second row in the answer in Figure 5 ("Manufacturing Process", ALL, ALL). Notice that all the different platform types show similar (though not exact) increase when going from 1992 to 1993. Hence, instead of listing them separately in the answer, we list a common parent of all these rows. The parent's ratio, shown as the RATIO column in Figures 3 and 5, is assumed to hold for all its children. The error incurred in making this assumption is indicated by the last column in the figures. In these experiments the error was calculated by taking a square root of the sum of squares of error of each detailed row. Notice that in both cases, this quantity is a small percent of the absolute difference of the values compared.

The compaction can be done in several different ways. The challenge is in choosing a method that on the one hand gives higher weightage to changes of larger magnitude but on the other hand allows summarization of rows which have almost similar changes. If we report everything at the most detailed level the error due to summarization is 0 but the coverage is also limited. Whereas if we aggregate heavily we can perhaps explain more of the change but the error due to summarization will also be high. We developed a information theoretic model for cleanly capturing these tradeoffs.

2.1 The model

Imagine a sender wishing to transmit the subcube C_b to a user (receiver) who already knows about subcube C_a. The sender could transmit the entire subcube C_b but that would be highly redundant because we expect a strong correlation between the values of corresponding cells in C_a and C_b. A more efficient method is to send a compact summary of the difference. The N-row answer $\mathcal{A}$ is such a summary. $\mathcal{A}$ consists of rows from not only detailed but also aggregated levels of the cube. With each aggregated row in the answer, we associate a ratio r that indicates to the user that everything underneath that row had the same ratio unless explicitly mentioned otherwise. For instance, in Figure 5 all Geography and Platforms for product category Health-Care are assumed to have a ratio of 118 except for those in 'United States' and Platform 'Multi-user mainframe'. We need to find $\mathcal{A}$ such that a user reconstructing C_b from C_a and $\mathcal{A}$ will incur the smallest amount of error. Intuitively, we can achieve this by listing rows that are significantly different than their parents and aggregating rows that are similar such that the error due to summarization is minimized. In information theory this error is characterized as the *number of bits* needed to transmit C_b to the user that already has C_a and $\mathcal{A}$. The number of bits are calculated using the classical Shannon's [CT91] information theorem which states that data $\vec{x}$ can be encoded using a model M that the sender and receiver have agreed on in $L(\vec{x}|M) \approx -\log \Pr[\vec{x}|M]$ bits where $\Pr[\vec{x}|M]$ denotes the probability of observing $\vec{x}$ given the model. In our case, data $\vec{x}$ is C_b, the model is C_a and $\mathcal{A}$ and the number of bits is calculated as follows:

```
For each detailed row $v$ in $C_b$
  If it already appears in $\mathcal{A}$
    the probability is 1 and cost($v$) = 0.
  Else,
    Find its most immediate parent $p \in \mathcal{A}$
    Let $r$ be the ratio associated with $p$
    The expected value $v_b$ of row $v$ in $C_b$ is $rv_a$.
    Assume a suitable probability distribution (as
    discussed in Section 2.1.1) centered around $rv_a$
    cost($v$) = $-\log \Pr[v_b|rv_a]$.
  Increment the total number of bits by cost($v$)
```

The ratio r associated with any $p \in \mathcal{A}$ is calculated as follows. When no child of p is in $\mathcal{A}$ then the ratio r is p_b/p_a. When a child of p is included in $\mathcal{A}$ we remove the contribution of the child to further refine the estimate of the ratio for children not included in $\mathcal{A}$.

The goal of the sender is then to pick an answer $\mathcal{A}$ that minimizes the total data transmission cost $-\log \Pr[C_b|C_a, \mathcal{A}]$.

In the above cost formulation we have ignored the cost of transmitting the answer set of size N since we assume that it is a user supplied parameter. We can easily include the cost of transmitting $\mathcal{A}$ by assuming a suitable model for transmitting each of its rows. One such model is to sum up the number of bits needed to transmit each attribute of each row as follows: For each attribute find the cardinality of the corresponding dimension of C_a. Since C_a is known to the user we only need to transmit an index of the required member. The number of bits needed for the index is: $\log(n_i + 1)$ where n_i denotes the cardinality of the ith dimension of cube C_a and "+1" is to add the possibility of sending "(All)". Once the number of bits needed for calculating an answer set of size N is known, we can select the value of N that leads to minimum total number of bits. Such an approach would be particularly useful when the user is at a loss about picking the value of the parameter N. In such a case, the algorithm could start with a suitably large value of N and finally chop the answer at a value of N for which the total cost is minimized.

2.1.1 Probability function

We use the probability distribution function as a means to convert the difference between the actual value v_b and the expected value rv_a into number of bits. In real-life it is hard to find a probability distribution function that the data follows perfectly. We found that it is not critical to choose a perfect model for the data. In choosing a function we only need to make sure that changes that are significant (contribute large amounts to the total) get more number of bits for the same ratio and rows with slightly different ratios can be easily summarized. For instance, if one row changes from 1 to 10 and another from 100 to 1000, the second change should be considered more interesting even though the ratio of change is the same. However, simply looking at magnitude is not enough because large numbers with even slightly different ratios would have large difference and it should still be possible to summarize them to enable inclusion of other changes that could be slightly smaller in magnitude but have much larger ratios.

For measures representing counts such as "total units sold" the Poisson distribution (with mean rv_a) provides a good approximation because we can think of each transaction as contributing a count of 1 or 0 to the measure independently of other transactions and thus forming a binomial distribution which we can approximate to Poisson when the number of transactions is large [Ham77]. For other measures, the normal distribution is often a safe choice and is used extensively in business analysis when more specific information about the data is lacking [Ham77].

2.1.2 Missing values

Another important issue closely tied to the choice of the probability distribution function is the treatment of missing values. These are cells that are empty in subcube C_a but not in C_b and vice versa. For example, for the cubes in Figure 2 there were products that were sold in 1990 but were discontinued in 1991. In most datasets we considered missing values occurred in abundance, hence it is important to handle them well.

One option is to treat a missing value like a very small constant close to zero. The trick is to choose a good value of the constant. Choosing too small a replacement for the missing values in C_a will imply a large ratio and even small magnitude numbers will be reported as interesting. Also rows with missing values in C_a and different values v_b in C_b will have different ratios. For instance, for a product newly introduced in

1991 all platforms will have a missing value in 1990. By replacing that value with the same constant we will get different ratios for different platforms and thus cannot easily summarize them into a single row.

A better solution is to replace all missing values as a constant fraction F of the other value i.e., if v_a is missing than replace v_a with v_b/F. The main issue then is picking the right ratio. Too large a ratio will cause even small values of v_b to appear surprising as shown in the example below for N = 3. When F = 10000 the answer picked is this.

(All)-	(All)-	749	719
Western Europe	(All)-	-	3
Western Europe	Multiuser UNIX	96	5

The second row in the answer (where v_a is missing and v_b is 3) is clearly an inferior choice compared to the row below:

United States	Multiuser Mainframe IBM	60	12

We handle missing values in a special manner. Whenever we are summarizing rows all of which have missing values of v_a we assume an ideal summarization and assign a cost of 0 due to error. Otherwise, we assume a suitably small value of v_a depending on the data distribution. For Poisson distribution we use the Laplace correction [Lap95] and assign $v_a = 1/(v_b + 1)$. For normal distribution we let $v_a = 0$ but use a variance of $v_b/2$.

2.2 Alternative formulations

We considered a few other alternatives in addition to the detail-N formulation discussed earlier and the information theoretic approach that we finally used.

The first alternative was based on the same structure of the answer but used a different objective function. It was formulated as a bicriteria optimization problem by associating each aggregated row with two quantities: a contribution term c that indicated what percent of the observed increase or decrease is explained by that row and a sum of squares of error term e that measured how much the ratio of its excluded children tuples differed from its ratio. The goal then was to find an answer with high total contribution and low total error. This alternative turned out to be unsatisfactory for several reasons. First, it was hard to formalize the goal. One way was to let the user place a bound on the error and choose an answer that maximizes total contribution within that error bound. But then it was unclear how the user would specify a good error bound. Second, choosing a row based on high contribution did not allow compact summarization of the form: "except for a single tuple t revenue for all others dropped uniformly by x%" because tuple t actually has a negative contribution and its role in the answer is to correct the error of its parent.

The second alternative we considered used a different structure of the answer. Instead of a flat set of rows it output a tiny decision tree where tuples with similar ratios were grouped within the same node of the tree. The problem with this approach is that the tree attempts to describe the entire data rather than pick out key summaries as we do in the present approach. This causes the description length to increase.

3 Algorithm

Finding an efficient algorithm for this problem is challenging because we wish to use the DIFF operator interactively and it is impossible to exhaustively solve in advance for all possible pair of cells that the user might possibly ask. Also to allow easy integration with existing OLAP engines we did not want to use special purpose storage structures or indices.

3.1 Greedy algorithm

The first cut greedy algorithm that we designed was as follows:

```
Compute all possible aggregates on concatenation of
  $C_a$ and $C_b$ i.e., cube($C_a|C_b$).
Initialize $\mathcal{A}$ with the top-most row of cube($C_a|C_b$).
This forms the first row of $\mathcal{A}$ as shown in Figure 3.
For i = 2 to N
    add to $\mathcal{A}$ the row from cube($C_a|C_b$) which
    leads to greatest reduction in number of bits.
```

There are two main problems with the above greedy algorithm. First, it requires as many passes of the data as the answer size N. Second, there are some common cases where it fails to get the optimal answer. We illustrate two such cases.

Consider the table in Figure 8 showing the revenues for different product categories for the data in Figure 1. We want to find the reasons for the revenue jumping from 9.2 million in 1992 to 287 million in 1994 for the category "System mgmt." . Suppose $N = 3$ for this example. The answer returned by the greedy algorithm is shown in Figure 10. According to the top-most row of the answer all Products except 'Storage Management' and 'Automated Operations' have a ratio of 16.3 between their values in 1992 and 1994. In Figure 9 we show the data at the next level of detail by drilling down to the Product dimension. We notice that for most products v_a is missing. Hence, the ratio of 16.3 introduces error for most Products like "Security Management" for which the ratio is almost infinity. A more forward looking algorithm could single out the only two products 'Performance Management' and 'Problem Management' for which the ratio is finite with the result that the final ratio of the top-row would be infinite as shown in Figure 11. With this solution, the total error is close to zero according to our model for handling missing values (Section 2.1.2). The greedy algorithm fails because it cannot predict the final ratio of the top-most row after removing the contribution of its divergent children tuples that finally

Platform	(All)		
Geography	(All)		
Product	(All)		
Sum of Revenue	Year		
Category	1992	1993	1994
Develop. tools	912.100	1287.070	1793.076
Info. tools	5.807	6.705	33.420
Office Apps	38.405	53.991	97.821
System mgmt.	9.233		278.612

Figure 8: Comparing revenues in 1992 and 1994 for Product Category 'System mgmt.'

Category	System mgmt.	
Platform	(All)	
Geography	(All)	
Sum of Revenue	Year	
Product	1992	1994
Automated Operations		59.108
Change & Configuration Mgmt.		27.905
Performance Management	5.515	66.565
Problem Management	3.718	35.897
Resource Accounting		11.421
Security Management		18.265
Storage Management		59.451

Figure 9: Details along different Products of 'System mgmt.' category.

PRODUCT	PLATFORM	GEOGRAPHY	YEAR_1992	YEAR_1994	RATIO
(All)-	(All)	(All)	9.8	160.1	16.3
Storage Management	(All)	(All)		59.5	
Automated Operations	(All)	(All)		59.1	

Figure 10: Answer returned by the greedy algorithm for the query in figure 8.

PRODUCT	PLATFORM	GEOGRAPHY	YEAR_1992	YEAR_1994	RATIO
(All)-	(All)	(All)		176.2	
Performance Management	(All)	(All)	5.5	66.6	12.1
Problem Management	(All)	(All)	3.7	35.9	9.7

Figure 11: A better solution to the difference query in Figure 8.

get included in the answer. This illustrates a failure of the algorithm even in the simple case of a single level of aggregation.

The next case illustrates the failure of the greedy algorithm even when the final ratio can somehow be magically predicted. This example is on a synthetic dataset with a two level hierarchy on a single dimension as shown in Figure 12. The topmost member m has optimal ratio 1 and the next intermediate level has three members m_1, m_2, m_3 with ratios 2, 1, 1 respectively. m_1 has one large child m_{11} with ratio 1 and several smaller children with ratio 2.5. Initially, $\mathcal{A}$ consists of just the topmost row with ratio 1. The reduction in the number of bits with adding m_1 to $\mathcal{A}$ is very small because of cancellations as follows: The predicted ratio for the children of m_1 changes from 1 to 2.0. While this helps the several small members with ratio 2.5, the benefit is canceled because for the large child m_{11} the error increases. The reduction in error with including any member with ratio 1 is 0. The reduction with including any of the other children of m_1 with ratio 2.5 is small but that is the best reduction possible and the final answer will include two of these children. However, the optimal answer consists of m_1 and its single divergent element m_{11} with ratio 1. This example indicates that another shortcoming of the greedy algorithm is its top-down approach to

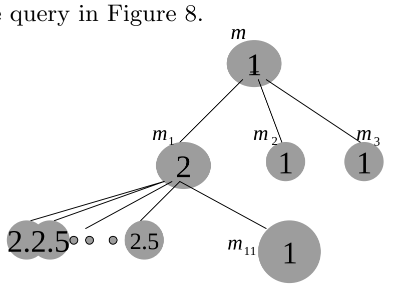


Figure 12: An example where greedy algorithm fails.

processing.

These two examples of the failure of the greedy solution should help the reader appreciate the difficult areas of the problem and it lead us to the dynamic programming algorithm described next.

3.2 Dynamic programming algorithm

The dynamic programming algorithm eliminates the above bad cases and is optimal under some assumptions elaborated next. We present the algorithm in three stages for ease of exposition. First, we discuss the case of a single dimension with no hierarchies. Next, we introduce hierarchies on that dimension and finally extend to the general case of multiple dimensions with hierarchies.

3.2.1 Single dimension with no hierarchies

In this case the subcube C_b consists of a one-dimensional array of T real-values. The basic premise behind the dynamic programming formulation is that we can decompose the set of tuples T into two subsets T' and $T - T'$, find the optimal solution for each of them separately and combine them to get the final answer for the full set T. Let $D(T, n, r)$ denote the total cost (number of bits) for T tuples, answer size n and final ratio of top-most row in $\mathcal{A}$ as r. Then,

$$D(T, n, r) = \min_{0 \leq m \leq n} (D(T - T', n - m, r) + D(T', m, r)) \quad (1)$$

This property enables us to design a bottom-up algorithm. We scan the tuples in $C_a|C_b$ at the detailed level sequentially while maintaining the best solution for slots from $n = 0$ to $n = N$. Assume that we magically know the best value of r. Let T_i denote the first i tuples scanned so far. When a new $(i + 1)$th tuple t_{i+1} is scanned we update the solution for all $(N + 1)$ slots using the equation above with $T' = t_{i+1}$. Thus, the solution for the slot n of the first $(i + 1)$th tuples is updated as

$$\begin{aligned} D(T_{i+1}, n, r) = \min \quad & (D(T_i, n - 1, r) + D(t_{i+1}, 1, r), \\ & D(T_i, n, r) + D(t_{i+1}, 0, r)) \end{aligned}$$

By the cost function in Section 2.1 $D(t_{i+1}, 1, r) = 0$ and $D(t_{i+1}, 0, r) = -\log \Pr[v_b|rv_a]$.

The best value of r is found by simultaneously solving for different guesses of r and refining those guesses as we progress. At start, we know (as part of the query parameters) the global ratio r_g when none of the tuples are included in the answer. We start with a fixed number R of the ratios around this value from $r_g/2$ to $2r_g$. We maintain a histogram of r values that is updated as tuples arrive. Periodically, from the histogram we pick $R - 1$ different r values by dropping up to N extreme values and using the average over the middle r values. We then select the R most distinct values from the R previous values and the $R - 1$ new values and use that to update the guess. When the algorithm ends, we pick the solution corresponding to that value of r which has the smallest cost. Thus, the final solution is:

$$D(T, N) = \min_{1 \leq i \leq R} D(T, N, r_i)$$

3.2.2 Single dimension with hierarchies

We now generalize to the case where there is a L level hierarchy on a dimension. For any intermediate hierarchy, we have the option of including in the answer the aggregated tuple at that level. When we include an aggregate tuple $agg(T)$, the ratio of all its children not in the answer is equal to the ratio induced by $agg(T)$ instead of the outer global ratio r. The updated cost equation is thus:

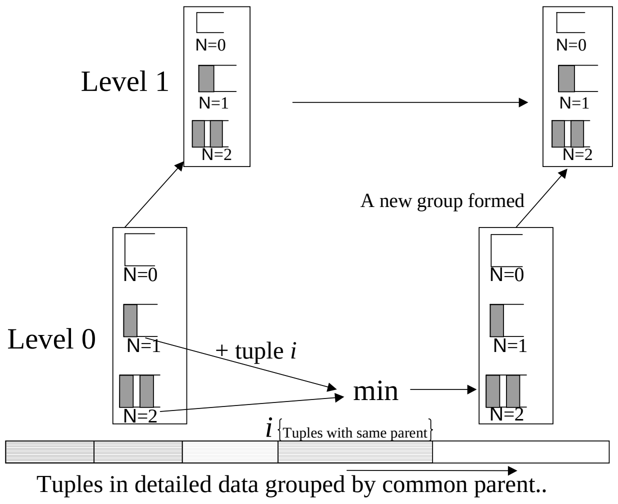


Figure 13: State of the dynamic programming algorithm for $N = 2$ and $L = 2$ and $R = 1$.

$$\begin{aligned} D(T, N, r) = \quad & \min(\text{cost } \textit{excluding} \text{ the aggregate tuple}, \\ & \text{cost } \textit{including} \text{ the aggregate tuple}) \end{aligned}$$

$$\begin{aligned} D(T, N, r) = \quad & \min(D(T, N, r) \text{ from equation (1)}, \\ & \min_{\forall r'}(D(T, N - 1, r') + agg(T))) \quad (2) \end{aligned}$$

The algorithm maintains L different nodes one for each of the L levels of hierarchy. Each node stores partial solutions for the $N + 1$ slots as described for the case of a single hierarchy. In Figure 13 we show an example state where the answer size $N = 2$ and the number of levels of the hierarchy $L = 2$. The tuples in $C_a|C_b$ are then scanned in an order such that all tuples within the same hierarchy appear together. The tuples are first passed to the bottom-most (most detailed) node of the hierarchy. This node updates its solution using Equation 1 until it gets a tuple that is not a member of the current hierarchy. When that happens it finds the final best solution using Equation 2, passes the solution to the node above it and re-initializes its state for the next member of the hierarchy. The next non-leaf node on getting a new solution from a node below it updates its solution using the same procedure. Thus, data is pipelined from one level to the next up the hierarchy. At each level we find the best solution for the group of tuples that have the same parent at that level of hierarchy. The final answer is stored in the top-most node after all the tuples are scanned.

Next we discuss how to choose values for the ratios for internal nodes. We need to allow for the possibility that a node's aggregate tuple may not be included. Instead *any* of its parent up to the root could become its immediate parent in the final answer. Hence, we need to solve for ratios of those parent tuples. Lacking any further information we start with a fixed set of ratios around the global ratio as in the single hierarchy case. When more tuples arrive at a node, we bootstrap towards a good choice of ratio as follows: Each node propagates downwards to its children node a current best guess of the ratio of its aggregate tuple. For tuples that are already through nothing can be done except re-evaluate costs with the new ratios but for subsequent tuples we get progressively better estimates.

Lemma 3.1 The above dynamic programming algorithm generates the optimal answer if the ratios guessed at each level are correct.

Proof. The proof of optimality follows directly from Equation 2 since the algorithm is a straight implementation of that equation with different ways of decomposing the tuples into subsets. The optimality of a dynamic programming algorithm is not affected by how and in how many parts we decompose the tuples as long as we can find the optimal solution in each part separately and combine them optimally. For $L = 1$ the decomposition is such that the new tuple becomes the T' in the equation and the size of T' is 1 always. For $L \geq 1$, we decompose tuples based on the boundaries of the hierarchy and T' consists of all children of the most recent value at that level of the hierarchy. ■

3.2.3 Multiple dimensions

In the general case we have multiple dimensions with one or more levels of hierarchies on each of the dimensions. We extend to multiple dimensions by preordering the levels of dimension and applying the solution of Section 3.2.1. The goal is to pick an ordering that will minimize the total number of bits. Intuitively, a good order is one where levels that show more similarity (in terms of ratio) are aggregated earlier. For example, for the queries in Figure 2 and Figure 4 on the data shown in Figure 1 we found the different levels of the platform dimension to be more similar than different levels of the product dimension. Therefore, we aggregated on platform first. We formalize this intuition by framing in the same information theoretic terms. For each level of each dimension, we calculate the total number of bits needed if all tuples at that level were summarized to their parent tuple in the next level. For each level l of dimension d we first aggregate the values to that level. Let T_{ld} denote the set of aggregated tuples at that level. We next estimate the error incurred if each tuple $t \in T_{ld}$ is approximated by the ratio induced by its parent tuple at level $l - 1$ as follows:

$$B_{ld} = \sum_{t \in T_{ld}} D(t, 0, r_{\text{parent(t)}})$$

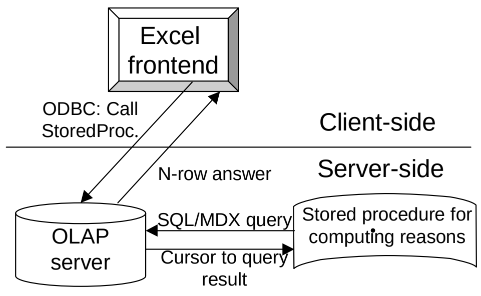


Figure 14: Our prototype.

Finally, we sort the levels of the dimension on B_{ld} with the smallest value corresponding to the lowermost level. Of course, in the final order the more detailed levels have to be below the less detailed level of the same hierarchy irrespective of the order specified by B_{ld}. The intuition behind this form of ordering is that, if the tuples within a level are similar, the parent tuple will be a good summary for those tuples and the number of bits needed to transmit those tuples to a receiver who already knows the parent tuple will be small. In contrast, if the children of a tuple are very divergent a larger number of bits will be needed to transmit them.

4 Implementation

4.1 Integrating with OLAP products

Our current prototype is based on IBM's DB2/UDB database (version 5.2) that we view as a ROLAP engine. We use SQL to access data such that partial processing is done inside the DBMS. In Figure 14 we show the architecture for our prototype. Our algorithm is packaged as a stored procedure that resides on the server side. The client (Microsoft Excel in our case) uses ODBC embedded in Visual Basic for invoking the stored procedure. The N row-answer is stored in the database and accessed by the client using ODBC. The stored procedure generates a giant SQL statement that subsets only the relevant part of the data cube. The query involves selecting the specified values at the specified aggregation level and sorting the data such that the stored procedure does not have to do any intermediate caching of the results. In Figure 15 we show the example SQL statement for the difference query in Figure 2. The data is assumed to be laid out in a star schema [CD97].

The stored procedure is a rather light-weight attachment to the main server. The indexing and query processing capability of the server is used to do most of the heavy-weight processing. Thus, the stored procedure itself does not use any extra disk space and the amount of memory it consumes is independent of the number of rows. It is $O(NLR)$ where N denotes the maximum answer size, L denotes the number of levels of hierarchy and R denotes the number of distinct

```
with subset(productId, platformId, year, revenue) as
   select productId, platformId, year, revenue from cube
   where geographyId = (select geographyId from geography where name = 'Rest of World')
with cube-A(productId, platformId, v_a, v_b) as
     select productId, platformId, revenue, 0 from subset
     where year = 1990
with cube-B(productId, platformId, v_a, v_b) as
     select productId, platformId, 0, revenue from subset
     where year = 1991
select prod_groupId, prod_categoryId, productId, plat_userId, plat_typeId,
   platformId, sum(v_a), sum(v_b)
   from (select * from cube-A) union all (select * from cube-B)
   group by prod_groupId, prod_categoryId, productId, plat_userId, plat_typeId, platformId
   order by prod_groupId, prod_categoryId, productId, plat_userId, plat_typeId, platformId
```

Figure 15: SQL query submitted to the OLAP server.

ratio values tried by the algorithm. This architecture is thus highly scalable in terms of resource requirements. We are also building a second prototype using the emerging OLE DB for OLAP [Mic98b] standard for integrating with any OLAP engine that supports the API.

4.2 Performance evaluation

In this section we present an experimental evaluation on our prototype to demonstrate the (1) feasibility of getting interactive answers on typical OLAP systems and the (2) scalability of our algorithm as the size and dimensionality of the cube increases.

All the experiments were done on a IBM PC with a 333 MHz Intel processor, 128 MB of memory and running Windows NT 4.0. We used the following datasets.

Software revenue data: This is a small dataset but is interesting because it is real-life data about the revenues of different software products from 1990 to 1994. We discussed this dataset earlier in Section 1.2 and repeat the schema here for convenience. The numbers within bracket denote the cardinality of that level.

Product	Platform	Geography	Year
Product name (67)	Platform name (43)	Geography (4)	Year (5)
Prod_Category (14)	Plat_Type (6)		
Prod_Group (3)	Plat_User (2)		

OLAP Council benchmark [Cou]: This dataset was designed by the olap council to serve as a benchmark for comparing performance of different OLAP products. It has 1.36 million total non-zero entries and four dimensions: Product with a seven hierarchy, Customer with a three level hierarchy, Channel with no hierarchy and Time with a four level hierarchy as shown in the figure below.

Product	Customer	Channel	Time
Code (9000)	Store (900)	Channel (9)	Month (17)
Class (900)	Retailer (90)		Quarter (7)
Group (90)			Year (2)
Family (20)			
Line (7)			
Divison (2)			

Grocery sales data: This is a demo dataset obtained from the Microsoft DSS product [Mic98a]. It has 250 thousand total non-zero entries and consists of five dimensions with hierarchies as shown below.

Store	Customer	Product	Promotion	Time
Name (24)	City (109)	Name (1560)	Media type (14)	Month (24)
State (10)	State (13)	Subcategory (102)		Quarter (8)
Country (3)	Country (2)	Category (45)		Year (2)
		Department (22)		
		Family (3)		

We establish the computational feasibility of answering online why queries. Unlike conventional data mining algorithms, we intend this tool to be used in an interactive manner hence the processing time for each query should be bounded. In most cases, although the entire cube can be very large, the subset of the cube actually involved in the processing is rather small. When that is not true we bound the processing time by limiting the level of detail from which we start. When the server is first initialized it collects statistics of the number of tuples at various levels of detail and uses that to determine the level of detail from which the processing should start.

The queries for our experiments are generated by randomly selecting two cells from different levels of aggregation of the cube. There are three main attributes of the workload that affect processing time:

- The number of tuples in the query result (size of $C_a|C_b$).
- The total number of levels in $C_a|C_b$ that determines the number of nodes (L) in the dynamic programming algorithm.
- The answer size N that determines the number

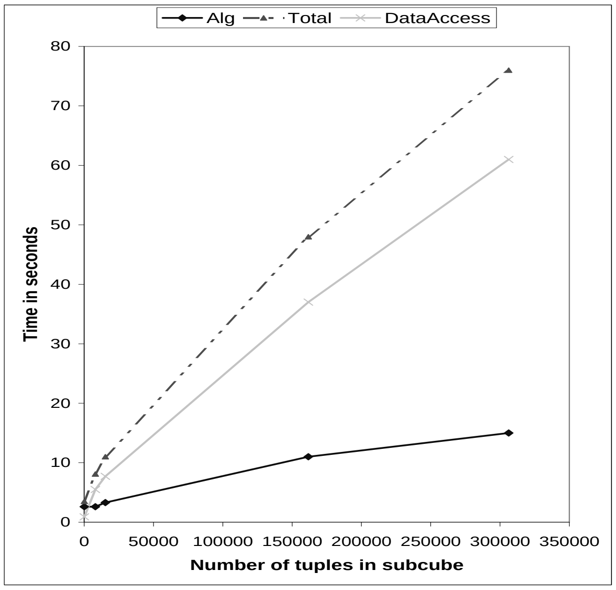


Figure 16: Total time taken as a function of subcube size

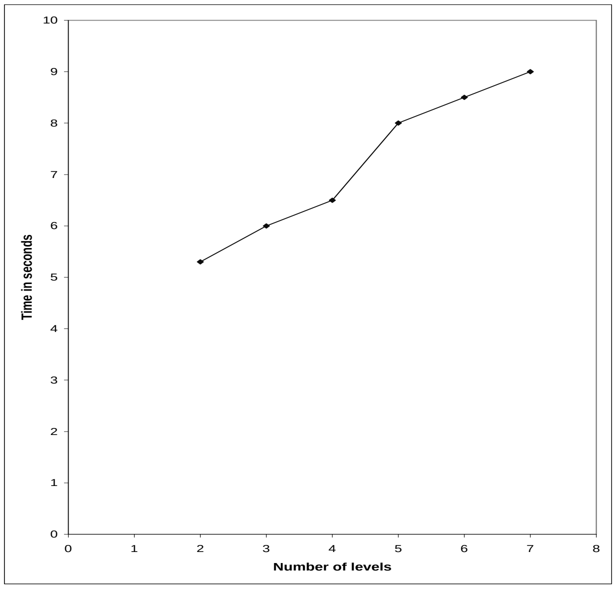


Figure 17: Total time taken versus number of levels in the answer

of slots per node. The default value of N in our experiments was 10.

We report on experimental results with varying values of each of these three parameters in the next three sections.

4.2.1 Number of tuples

We chose ten arbitrary queries within two levels of aggregation from the top. In Figure 16 we plot the total time in seconds for each query sorted by the number of non-zero tuples in the subcube $(C_a|C_b)$.

We show three graphs: The first one denotes the data access time which includes the time from the issue of the query to returning the relevant tuples to the stored procedures. The second curve denotes the time spent in the stored procudure for finding the answer. The third curve denotes the sum of these two times. From Figure 16 we can make the following observations:

- The total time taken for finding the answer is less than a minute for most cases. Even for the subcube with quarter million entries the total time is only slightly over a minute.
- Only a small fraction $< 20\%$ of the total time is spent in the stored procedure that implements the DIFF operator. The majority of the time is spent in subseting the relevant data from the database and passing to the stored procedure. This implies that if we used a server better optimized for answering OLAP queries the processing time could be even further reduced.
- The subset of the data actually relevant to the query is often very small even for fairly large sized datasets. For example, the OLAP-benchmark dataset has 1.37 million total tuples but the largest size of the subcube involving two comparisons along the month dimension is only 81 thousand for which the total processing time is only 8 seconds.

4.2.2 Number of levels

In Figure 17 we show the processing time as a function of the number of levels of aggregation. As we increase the number of levels, for a fixed total number of tuples, we expect the processing time to increase, although at a slower than linear rate because significantly more work is done at lower levels than higher levels of the node. The exact complexity depends also on the fanout of each node of the hierarchy. In Figure 17 we observe that as the number of levels is increased from 2 to 7 the processing time increases from 6 to 9 seconds for a fixed query size of 70 thousand tuples. This is a small increase compared to the total data access time of 40 seconds.

4.2.3 Result size N

In Figure 18 we show the processing time as a function of the answer size N for two different queries: query 1 with 8 thousand tuples and query 2 with 15 thousand tuples. As the value of N is increased from 10 to 100 the processing time increases from 2.7 to 6 seconds for query 1 and 3.1 to 9 seconds for query 2. When we add the data access time to the total processing time, this amounts to less than a factor of 2 increase in total time. The dynamic-programming algorithm has a $O(N^2)$ dependence on N but other fixed overheads

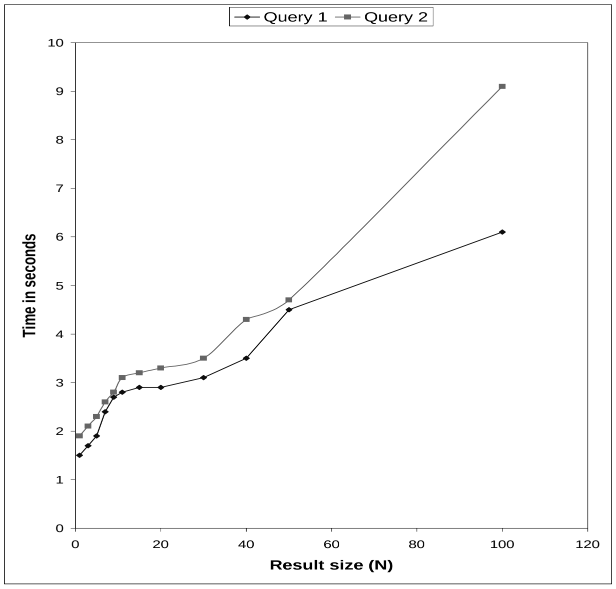


Figure 18: Total time taken as a function of result size(N)

dominant the total processing time more than the core algorithm. That explains why even when we increase N from 1 to 100, the processing time increases by less than a factor of 5.

5 Conclusion

In this paper we introduced a new operator for enhancing existing multidimensional OLAP products with more automated tools for analysis. The new operator allows a user to obtain in one step summarized reasons for changes observed at the aggregated level.

Our formulation of the operator allows *key* reasons to be conveyed to the user using a very small number of rows that (s)he can quickly assimilate. By casting in information theoretic terms, we obtained a clean objective function that could be optimized for getting the best answer. We designed a dynamic programming algorithm that optimizes this function using a single pass of the data and consuming very little memory. This algorithm is significantly better than our initial greedy algorithm both in terms of performance and quality of answer.

We prototyped the operator on the DB2/OLAP server using an excel front-end. Our design enables most of the heavy-weight processing involving index-lookups and sorts to be pushed inside the OLAP server. The extension code needed to support the operator does relatively smaller amount of work. Our design goal was to enable interactive use of the new operator and we demonstrated through experiments on the prototype. Experiment using the industry OLAP benchmark indicate that even when the subcubes defined by the DIFF query includes a quarter million tuples we can process them in a minute. Our experiments also show the scalability of our algorithm as the number of tuples, number of levels of hierarchy and the answer size increases.

In future we wish to design more operators of this nature so as to automate more of the existing tedious and error-prone manual discovery process in multidimensional data.

References

[Arb] Arbor Software Corporation, Sunnyvale, CA. *Multidimensional Analysis: Converting Corporate Data into Strategic Information.* `http://www.arborsoft.com`.

[CD97] S. Chaudhuri and U. Dayal. An overview of data warehouse and OLAP technology. *ACM SIGMOD Record*, March 1997.

[Cod93] E. F. Codd. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. Technical report, E. F. Codd and Associates, 1993.

[Cor97a] Cognos Software Corporation. Power play 5, special edition. `http://www.cognos.com/powercubes/index.html`, 1997.

[Cor97b] International Data Corporation. `http://www.idc.com`, 1997.

[Cou] The OLAP Council. The OLAP benchmark. `http://www.olapcouncil.org`.

[CT91] Thomas M Cover and Joy A Thomas. *Elements of Information Theory.* John Wiley and Sons, Inc., 1991.

[Dis] Information Discovery. `http://www.datamine.inter.net/`.

[Ham77] M. Hamurg. *Statistical analysis for decision making.* Harcourt Brace Jovanovich, Inc, New York, 1977.

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. of the 21st Int'l Conference on Very Large Databases*, Zurich, Switzerland, September 1995.

[Lap95] P-S Laplace. *Philosophical Essays on Probabilities.* Springer-Verlag, New York, 1995. Translated by A. I. Dale from the 5th French edition of 1825.

[Mic98a] Microsoft corporation. *Microsoft decision support services version 1.0*, 1998.

[Mic98b] Microsoft Corporation, `http://www.microsoft.com/data/oledb/olap/spec/`. *OLE DB for OLAP version 1.0 Specification.*, 1998.

[SAM98] Sunita Sarawagi, Rakesh Agrawal, and Nimrod Megiddo. Discovery-driven exploration of OLAP data cubes. In *Proc. of the 6th Int'l Conference on Extending Database Technology (EDBT)*, Valencia, Spain, 1998. expanded version available from `http://www.almaden.ibm.com/cs/quest`.

[Sof] Pilot Software. Decision support suite. `http://www.pilotsw.com`.