iDiff: Informative summarization of differences in multidimensional aggregates*

Sunita Sarawagi
Indian Institute of Technology, Bombay
sunita@it.iitb.ernet.in

Abstract

Multidimensional OLAP products provide an excellent opportunity for integrating mining functionality because of their widespread acceptance as a decision support tool and their existing heavy reliance on manual, user-driven analysis. Most OLAP products are rather simplistic and rely heavily on the user's intuition to manually drive the discovery process. Such ad hoc user-driven exploration gets tedious and error-prone as data dimensionality and size increases. Our goal is to automate these manual discovery processes. In this paper we present an example of such automation through a iDiff operator that in a single step returns summarized reasons for drops or increases observed at an aggregated level.

We formulate this as a problem of summarizing the difference between two multidimensional arrays of real numbers. We develop a general framework for such summarization and propose a specific formulation for the case of OLAP aggregates. We design an efficient dynamic programming algorithm that requires only one pass of the data and uses a small amount of memory independent of the data size. This allows easy integration with existing OLAP products. Our prototype has been tested on the Microsoft OLAP server, DB2/UDB and Oracle 8i. Experiments using the OLAP benchmark demonstrate (1) scalability of our algorithm as the size and dimensionality of the cube increases and (2) feasibility of getting interactive answers with modest hardware resources.

Keywords: Multidimensional databases, OLAP, OLAP-mining integration, difference mining, data summarization, advanced aggregates.

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

*An abridged version of this paper appeared in [Sar99] there are navigational operators like select, drill-down, roll-up and pivot conforming to the multi-dimensional 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. Mining technology can play a fitting role in improving the state of these products. The huge size of the data and the rich star-like structure based on dimensions and hierarchies make it possible to integrate mining tools in useful and innovative ways. Existing tools like classification, clustering, association rules and time series analysis are starting to be deployed, albeit sparingly [IDA99, Tho98, Han98, Ger96, DBM98, Inc96]. Recent work in this direction attempt to enhance OLAP products with known mining primitives. For instance, Information Discovery Inc [Dis] and Cognos [Cor97a] white papers discuss how classifiers can be used to understand factors affecting measures like profit. The profit figures are divided into some fixed number of categories: "low", "medium" and "high" and classified as a function of attributes in the dimensions, hierarchies and any other information associated with the dimensions. Similarly, clustering has useful applications in the OLAP domain. In most corporate data cubes the individual customer names lead to high sparsity. Hence, normally the customer names are aggregated to a higher level like customer location. However, the customer's location might have little to do with the buying patterns of customers. More meaningful hierarchies can be defined by applying a hierarchical clustering algorithm on customers based on their buying patterns and any other attributes associated with the customer. Han et al [HF95] discuss how association rules at multiple levels of aggregation can be used to find progressively detailed correlation between members of a dimension. Time is a critical dimension in OLAP data cubes and time series analysis methods are directly relevant. Some vendors have support for querying for patterns like "spikes", "gradients", "periodicity" and "outliers" along time. Also, most OLAP products provide support for traditional time series analysis functions like forecasting.

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. We are developing a suite of these operators in the context of the $\mathbf{i^3}$ (for "eye-cube") project [SS00]. One of these is the iDiff operator described here. The goal for this operator is to automate an 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 company's annual sales 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 the new iDiff operator that 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.

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.

Platform	(All)				
Product	(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.

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.

1.1 Contents

We first demonstrate the use of the new iDiff 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 the "iDiff" operator. The result as shown in Figure 3 is a list of at most 10 rows (the number 10 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

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.I	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".

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 iDiff operator instead of doing multiple drill downs and selection steps on the huge amount of

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 Euro	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 DetailSort approach.

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 g_a and g_b in the data cube and wishes to know why one is greater or smaller than the other. Both g_a and g_b are formed by aggregating values along one or more dimensions. Let C_a and C_b denote the two subcubes formed by expanding the common aggregated dimensions of g_a and g_b. For example, in Figure 2 g_a denotes the revenue for the cell (All platforms, All products, 'Rest of world', 1990) and g_b denotes the total revenue for the 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

A simple approach is to show the large changes in detailed data sorted by the magnitude of the change (DetailSort 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 DetailSort approach explains only 900 out of the total difference of about 5000. In contrast, with the answer in Figure 5 we could explain more than 4500 of the total difference. The compaction is achieved by 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

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.

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.

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 the corresponding cells in C_a and C_b. A more efficient method is to send a compact summary of the difference where cells with similar changes are grouped together. There are several methods of summarization arising out of different choices of grouping criteria, error measures, group format and objective function. The **grouping criteria** determines how we measure the commonality in the relationship between corresponding values v_a and v_b in C_a and C_b respectively. For instance, we could summarize cells based on their having the same absolute difference between v_a and v_b or their having the same ratio. Each collection of cells having the same value of this relationship function will be grouped together and represented as a single summarized unit. A closely related issue is the **error function** that determines how much error is incurred in reconstructing v_b from v_a and the summarized answer. For instance, if "ratio" is used to characterize relationship between cells, what is the error function for cells that have ratio slightly different from the ratio of its group. The **grouping format** determines what subset of cells are considering for grouping together. Grouping arbitrary collection of cells is neither easy to interpret nor computationally feasible. Finally, the **objective function** that is used to characterize the best solution. Some example objective functions are minimize total error given a fixed limit on the size of the summary or minimize summary size given a limit on the maximum error. We elaborate on each of these four options and our chosen method in each.

2.1.1 Grouping criteria

The grouping criteria determines how we measure similarity of the relationship between corresponding values in cubes C_a and C_b. This has to be determined based on external intuition about how we expect values to be related. For instance, if C_a and C_b refer to sales in 1998 and 1999 respectively for different products, how do we expect sales across different Products to be related? For OLAP datasets we found that "ratio" is a better choice than "difference". Different products are likely to have the same percentage difference rather than the same absolute difference. In the final answer, each group of cells is associated with a ratio value that best characterizes the members of its group.

2.1.2 Error function

This measures the error due to summarization i.e., the gap between the actual value v_b in cube C_B and the expected value derived from the summary statistic of its group and the value v_a of cube C_A. Example error functions are squared difference, absolute difference, chi-squared error, log likelihood, KL distance and so on. In choosing a function we only need to make sure that changes that are significant (contribute large amounts to the total) get higher error for the same ratio and cells 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 OLAP data, if the ratio of a group of cells is r, then the expected value of cell in cube C_B is rv_a. We measure the error $\mathrm{Err}(v_a, v_b, r)$, between the actual value v_b and the expected value rv_a as follows:

$$\mathrm{Err}(v_a, v_b, r) = (v_b - rv_a) \log \frac{v_b}{rv_a} \tag{1}$$

For the same ratio of actual and expected values, the above function gives higher weightage to larger differences and for the same absolute difference the function gives high weightage to larger ratio. Thus both the magnitude and ratio of change are important. Also, notice that the error is always a positive value. We found that the commonly used squared or absolute error were not suitable. Both these functions give high weightage to large numbers — a smaller ratio for a larger number appeared more surprising than a much larger ratio for a smaller number. This tended to pick large outliers from the most detailed level as the reasons.

In general the formulation allows any user-defined error function but computationally it is more efficient to have one that is incrementally computable. Incremental computability requires that given a set of cells we should be able to compute a small number of aggregate functions on v_a and v_b such that for any value of the group ratio r we can calculate the sum of the errors from these tuples. For example for our chosen error function in Equation 1 we can rewrite as follows:

$$(v_b - rv_a) \log \frac{v_b}{rv_a} = v_b \log \frac{v_b}{v_a} - v_b \log r - rv_a \log \frac{v_b}{v_a} + rv_a \log r \tag{2}$$

When aggregating tuples we compute four sums $S_1 = \sum v_b \log \frac{v_b}{v_a}$, $S_2 = \sum v_b$ and $S_3 = \sum v_a \log \frac{v_b}{v_a}$ and $S_4 = \sum v_a$ Now, given any r we can compute the total error as: $S_1 - S_2 \log r +$

$S_3 r + S_4 r \log r$. Similar decompositions are possible for the square error function. Error functions like the L_1 distance defined as $|v_b - rv_a|$ are not incrementally computable.

Missing values Another important issue when defining error functions is how missing values are handled. 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 and vice versa. The main issue then is picking the right ratio. Too large a ratio will cause even small values of v_b to appear surprising. Therefore, we cannot set F to any arbitrarily large value. However, the danger of setting it to a moderately large value is that the data itself might have ratios between values that are much larger than F. Without looking at the data, we found it really hard to pick an absolute value of F that will work well for all cases.

We handle missing values in a special manner. Whenever we are summarizing rows all of which have missing v_a values or all have missing v_b values we assume an ideal summarization and assign a cost of 0 due to error. Otherwise, we have a case of trying to group together tuples where some of them have missing v_a, some have missing v_b and the rest have no missing values. We treat each kind separately in summing the total error. Let us assume that r is the group ratio. Then the error in the first kind where the value of v_a is missing is calculated assuming that the fraction $\frac{v_b}{v_a}$ is $\max(F, r + 1/F)$ otherwise $v_a = 0$. This ensures that the ratio due to the missing value is always greater than the maximum ratio of the group. The error in the second kind where value of v_b is missing is calculated assuming that fraction $\frac{v_b}{v_a}$ is $\min(1/F, r)$ otherwise $v_b = 0$. This ensures that the ratio due to the missing value is never greater than the group ratio. We are still left with having to choose F but the danger of a bad choice is significantly reduced.

2.1.3 Group Format

The next issue is what subset of cells are considered for grouping together. We wanted to choose a representation that would be easy for a user to interpret. Grouping together arbitrary subset of members along a dimension is not meaningful because dimension normally are categorical. In most cases dimensions have hierarchies and that yields natural boundaries for grouping. A group is represented along each dimension as a single value from any level of the hierarchy. The answer consists of a collection of such groups. These groups can be overlapping with a cell always binding to the most detailed group in the answer. For instance, in Figure 5 the third row denoting tuples with Product name "Other Vertical Apps" overlaps with tuples with the same product name under "United States" geography and "Unix S." platform. Therefore, all tuples belonging to the second group have been implicitly removed from the first group.

There are other alternatives to groupings. For instance, instead of a flat set groups we can

$\mathcal{A}$	Answer transmitted to user
N	User limit on number of tuples in $\mathcal{A}$
$g_a(g_b)$	First (second) aggregated quantity pointed to by user
$C_a(C_b)$	Subcubes formed by expanding common dimensions of g_a and g_b
r_g	Global ratio observed by user $= g_b/g_a$
t	A tuple in the subcube
$t_a(t_b)$	Measure part of tuple t in subcube $C_a(C_b)$
$\text{Err}(t_a, t_b, r)$	Error of tuple t with measures t_a and t_b when expected ratio is r
$\text{Err}(t, r)$	Error of tuple t with measures t_a and t_b when expected ratio is $r = \text{Err}(t_a, t_b, r)$

Table 1: Notations

output a tree where tuples with similar ratios were grouped within the same tree node. 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.

2.1.4 Objective function

. We allow a choice of two different objective functions. In the first case, the user specifies a fixed limit N on the number of rows in the final answer and the objective is to minimize total error. Whenever two different answers have the same total error, the smaller answer is chosen. In the second case, the user specifies a fixed limit e_{max} on the error of each cell and the objective is to find the smallest possible answer that meets this limit for each cell. Whenever two answers have the same size, the one with the smallest total error is chosen. Other formulations, for example, ones which automatically try to choose the right value of N based on some MDL-ish[CT91] formulation is perhaps also possible. Our attempts at such automatic choice of parameters failed because there did not appear any principled way to measure the smallest possible description length of an answer.

2.2 Other interesting formulations

In the earlier section we presented the general framework for reporting the difference between two aggregated values and also discussed the specific formulation we chose. In this section we discuss how this general framework can be used to answer some other interesting query forms. We present some common scenarios:

One-sided errors Sometimes, a user might be interested in seeing only rows with significant increase if the top-level row had an increase and similarly rows with significant drop if the top-level row had a drop. The only time a row with opposite change would be reported is when an intermediate parent (other than the top-level) is also present in the answer. We can capture this requirement by modifying the error function to have a value of “0” if the sign of the change is opposite to that of the topmost parent and there are no other intermediate parents of the row. Otherwise it is the same as before. We call these **one-sided error functions.**. This error function can also be made to be incrementally computable even though from the definition it may not be immediately apparent. The error of a detailed tuple depends on whether an intermediate parent of the tuple is included or not. Therefore, we need to maintain some more aggregates in addition to the four presented in Equation 2. We maintain a set of four more sums that are otherwise exactly the same as the first four but only add tuples where the sign of the change is the same as the toplevel parents. This enables us to use the general algorithm (discussed in Section 3) that depends on the incremental computability of the error function.

Non-additive aggregate functions In the discussion so far, the underlying assumption was that the aggregate function based on which tuples are aggregated is sum of a single measure, for example total sales and total volume. When the quantity displayed is: percentage increase in sales or percentage profit or average sales than this assumption is not true. In fact, what we have is a ratio of two of these additive aggregate functions. We discuss how some of these composite aggregate functions can be transformed to our formulation through a few scenarios:

- Why is the percent increase in sales for product A higher than for product B this year? A direct approach would be to treat subcube C_a as the percent increases of product A and subcube C_b as the percentage increases in product B. We do not use this approach because the percent increase at detailed levels are not added together to aggregate to higher levels, hence the error functions as defined in Equation 1 fails to give higher weightage to larger contributors to differences at aggregate levels. Instead, the following transformation works. We identify four subcubes: the subcubes for product A and product B this year and the last. We assume that the user knows about the first three subcubes. Accordingly, the user expects the unknown cells in subcube for product B this year to have increased by the same percent as product A. The answer $\mathcal{A}$ corrects the error in making this assumption. Hence, subcube C_a is the sales of product B last year scaled by the percentage increase of product A and subcube C_b is the sales of product B this year.
- Why is the profit percent in region-A higher than in region-B? For answering this query, again instead of comparing directly the two profit percents which are not additive we identify the constitute additive aggregates: sales and cost price. We then transform the cube to define subcube C_a as the expected sales in region B derived from the costs in region B and the profits in region A and define subcube C_b to be the actual sales in region B.

3 Algorithm

Finding an efficient algorithm for this problem is challenging because we wish to use the 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. We present algorithms for the two objective functions discussed in Section 2.1.4: minimizing total error given fixed answer size N and minimizing answer size given a limit on maximum error of inidividual tuples.

3.1 Minimizing total error given fixed answer size N

In [Sar99] we showed how top-down greedy methods can be sub-optimal. We elaborate our final algorithm based on dynamic programming that is optimal under certain conditions and is better than the greedy algorithm both in terms of performance and quality of the answer. 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.

3.1.1 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,

$$D(T, n, r) = \min_{0 \leq m \leq n} (D(T - T', n - m, r) + D(T', m, r)) \tag{3}$$

This property enables us to design a bottom-up algorithm. Let $C_a|C_b$ denote the tuples from C_a and C_b where the corresponding cells have been joined to get the measures from both C_a and C_b. 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 is scanned we update the solution for all $(N + 1)$ slots using the equation above with $T' = t$. Thus, the solution for the slot n of the first $(i + 1)$th tuples is updated as

$$D(T_{i+1}, n, r) = \min \quad (D(T_i, n - 1, r) + D(t, 1, r), D(T_i, n, r) + D(t, 0, r)) \tag{4}$$

By the cost function in Section 2.1 $D(t, 1, r) = 0$ and $D(t, 0, r) = \text{Err}(t_a, t_b, r)$ (refer notation in Table 1).

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:

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

3.1.2 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. Thus the set of tuples T consists not only of detailed tuples but also their parents at higher levels of the hierarchy. When we include a tuple $agg(T')$ which is a parent of all tuples T', then the default ratio of all its children T' 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:

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

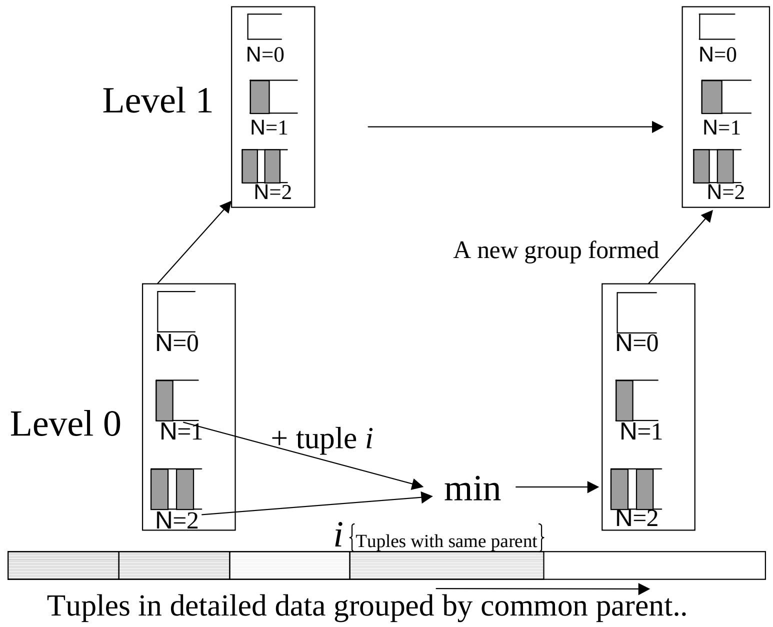

Figure 8: State of the dynamic programming algorithm for $N = 2$ and $L = 2$ and $R = 1$.

$$D(T' \cup agg(T'), n, r) = \min(D(T', n, r), D(T', n-1) + agg(T)) \tag{5}$$

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 8 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 3 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 5, 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.

We now 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 5 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. ∎

3.1.3 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 pre-ordering the levels of dimension and applying the solution of Section 3.1.1. The goal is to pick an ordering that will minimize the total error. Intuitively, a good order is one where levels with higher similarity amongst its members 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. For each level of each dimension, we calculate the total error if all tuples at that level were summarized to their parent tuple in the next level. If the tuples within a level are similar, the parent tuple will be a good summary for those tuples and the error due to summarization will be small. In contrast, if the children of a tuple are very divergent the error in grouping them via a single parent tuple will be large. Let T_{ld} denote the set of aggregated tuples at level l of dimension d. We next estimate the error incurred if each tuple $t \in T_{ld}$ is approximated by the ratio induced by its parent tuple at the next higher levels as follows:

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

Finally, we sort the levels of the dimension on B_{ld} with the smallest value corresponding to the lowermost level.

This computation of the similarity between members could be done off-line using data of the entire cube and a fixed ordering of dimensions stored or it could be done dynamically for each query for the subset actually involved in the query.

3.2 Minimizing answer size given fixed error limit

In this case, the user provides a limit e_{max} on the maximum error between the expected and actual values of any cell and the objective is to provide the smallest length summary that meets this error limit. This is an easier problem to solve than the previous one where the answer size is limited. We present the algorithm in the same three stages as in the previous case.

3.2.1 Single dimension with no hierarchies

Let $N(T, r, e_{max})$ denote the optimal answer size for a set of tuples T when r denotes final ratio of the parent of these T tuples. This optimal answer with a fixed value of r can be easily found as follows: Scan all tuples from T. If the error $\mathrm{Err}(v_b, rv_a)$ is greater than e_{max}, include the tuple in the final answer. The optimal answer $N(T, e_{max})$ corresponds to the r with the smallest final answer size.

$$N(T, e_{max}) = \min_r N(T, r, e_{max})$$

Values of r are chosen in exactly the same way as for the previous solution. The only difference with the previous solution is that we do not need to maintain separate solutions for different values of N.

3.2.2 Single dimension with hierarchies

Consider a set of tuples T that are all children of an internal node of the hierarchy. Let $agg(T)$ denote the parent of these T tuples. Our goal is to find the optimal solution $N(T \cup agg(T), r, e_{max})$ where r denotes the default ratio of any tuple not included in the final solution. There are two options for the solution corresponding to whether $agg(T)$ is included in the final answer or not. If $agg(T)$ is not present in the final solution the error of all tuples not included in the final solution is derived based on the external ratio of r. If $agg(T)$ is included in the final solution the external ratio$r$ has no effect on the error due to tuples in T. The best solution is the smaller of these two possibilities. Thus

$$N(T \cup agg(T), r, e_{max}) = \min(N(T, r, e_{max}), N(T, e_{max}) + 1$$

An interesting corollary of this solution is that the optimal answer size $N(T, r, e_{max})$ for different r values cannot differ by more than one. This is because if we have a value of r, r_{min} that gives the smallest answer size N_{min}, then the solution for other values of r can be easily formed by the N_{min} tuples and the aggregate tuple $agg(T)$ with overall ratio r_{min}.

3.2.3 Multiple dimensions

The case of multiple dimensions can be handled exactly the same way as fixed N as discussed in Section 3.1.3.

3.3 Level pruning

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. We found the probabilistic counting method proposed in [FM85] (also applied in the cube context in [SDNR96]) to be very effective in estimating the sizes of different levels in just one pass of the detailed data. In all five real-life datasets we tried, the estimate was within 10% of actual for all levels. For individual queries, the size of the relevant levels of aggregation of the subcube can be found by scaling the estimated total size by the the selectivity of the subcube with respect to the actual cube. The selectivity is estimated as reciprocal of the size of the level corresponding to the constant values defining the subcube. For instance, in the diff query in Figure 2 the selectivity is estimated as the reciprocal of the estimated size of level Geography×Year. These estimates are used to pick the level where the system expects to complete within a specified threshold. The rest of the processing remains the same.

Sometimes a user might want a continuously refining solution that can be stopped any time and still yield a reasonable solution. This would be directly supported if our solution proceeded top-down. Unfortunately, top-down processing may not produce optimal answers. An additional challenge is that, aggregated rows give little clue about the worth of drilling down from them further. We present the algorithm for the two cases of fixed N and fixed error limit.

3.3.1 Fixed N

We propose the following algorithm. First, find the best bottom-up solution $D(T_f, n)$ starting from tuples T_f that are all aggregated at level l_f. Value of l_f is chosen so as to be within the user's time constraints as discussed above. This yields the first version of the answer. Choose the next level of detailed l_s that can be processed within the next time threshold that the user is willing to tolerate. We will merge the best answer of tuples between layers l_s and l_f with the first version answer. Let T_{sf} denote the set of tuples starting from aggregate level s and going upto level f. For each value of n, the selected tuples in the final answer $D(T_f, n)$ induce a default ratio for each tuple at level s. Let R_n denote these ratios. Find the best solution $D(T_{sf}, n, R_m)$ for different values of m and n. We merge this solution to the best solution starting from level l_f as follows:

$$D(T_s, n) = \min_{0 \leq m \leq n} (D(T_f, m) + D(T_{sf}, n - m, R_m))$$

3.3.2 Fixed error limit e_{max}

The solution for this case is very similar to the solution above except that there is no need to find the solution for different values of n.

4 Implementation

4.1 Integrating with OLAP products

The DIFF operator forms part of the $\mathbf{i^3}$ system that is a suite of operators for enhancing OLAP products with mining primitives. The system is architected in three tiers where the topmost tier is a java frontend based on the Swing API, the bottom layer is the OLAP data source and the middle layer is the main $\mathbf{i^3}$ server. Our current prototype has been tested and integrated with three OLAP data sources: Microsoft OLAP server, Oracle 8i and IBM DB2/UDB (version 6.1). For the ROLAP engines (Oracle and DB2) we use SQL via ODBC to access data and for the Microsoft OLAP server we use MDX via OLE DB for OLAP [Mic98]. In response to a user query, the DIFF operator dynamically generates the correct SQL or MDX query and submits it to the OLAP data source. The query involves selecting the specified values at the specified aggregation level and sorting the data. Thus there is no intermediate caching to be done at the $\mathbf{i^3}$ server. In Figure 9 and

```
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)), Platform p, Product r
   where p.plat_Id = platformId and r.prod_Id = productId
   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 9: SQL query submitted to the OLAP server for the iDiff query in Figure 2.

```
with subset(productId, geographyId, platformId, year, revenue) as
   select productId, geographyId, platformId, year, revenue from cube
   where productId IN (select productId from Products where prod_category = 'Vertical Apps')
with cube-A(productId, geographyId, platformId, v_a, v_b) as
     select productId, geographyId, platformId, revenue, 0 from subset
     where year = 1992
with cube-B(productId, geographyId, platformId, v_a, v_b) as
     select productId, geographyId, platformId, 0, revenue from subset
     where year = 1993
select productId, geographyId, plat_userId, plat_typeId,
   platformId, sum(v_a), sum(v_b)
   from ((select * from cube-A) union all (select * from cube-B)), Platform p
   where p.plat_Id = platformId
   group by productId, geographyId, plat_userId, plat_typeId, platformId
   order by productId, geographyId, plat_userId, plat_typeId, platformId
```

Figure 10: SQL query submitted to the OLAP server for the iDiff query in Figure 4.

10 we show the example SQL statements for the difference queries in Figures 2 and 4 respectively. The data for this query is assumed to be laid out in a star schema [CD97].

The middle **i³** layer is a rather light-weight attachment to the OLAP server. The indexing and query processing capability of the server is used to do most of the heavy-weight processing. Thus, the diff algorithm itself does not use any extra disk space. For the case of fixed N, 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 ratio values the algorithm uses. This architecture is thus highly scalable in terms of resource requirements.

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 PC with a 333 MHz Intel processor, 128 MB of memory, running Windows NT 4.0 and DB2 UDB as the OLAP data source. The OLAP Council benchmark [Cou] is used for the experiments. 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)			

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 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 11 we plot the total time in seconds for each query sorted by the number of non-zero tuples in the subcube ($C_a|C_b$).

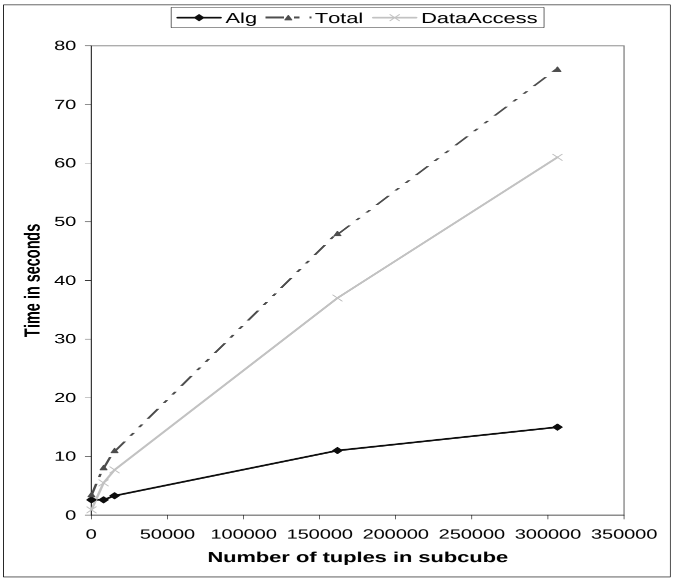

Figure 11: Total time taken as a function of subcube size

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 procedure for finding the answer. The third curve denotes the sum of these two times. From Figure 11 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 iDiff 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 12 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 12 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.

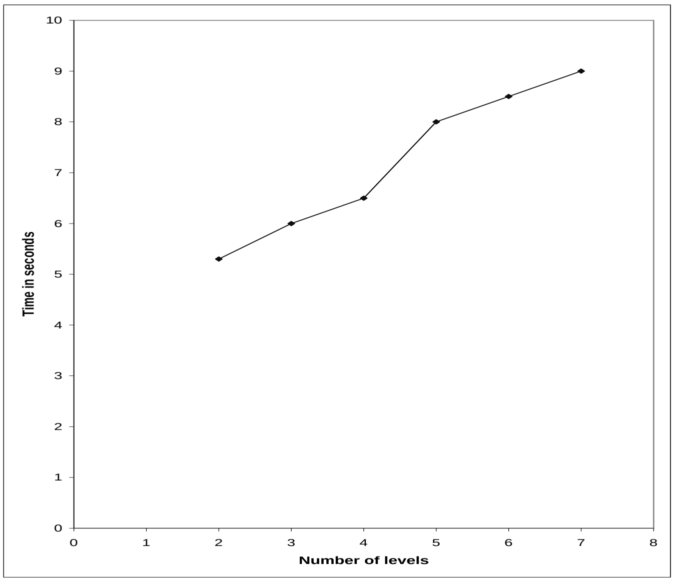

Figure 12: Total time taken versus number of levels in the answer

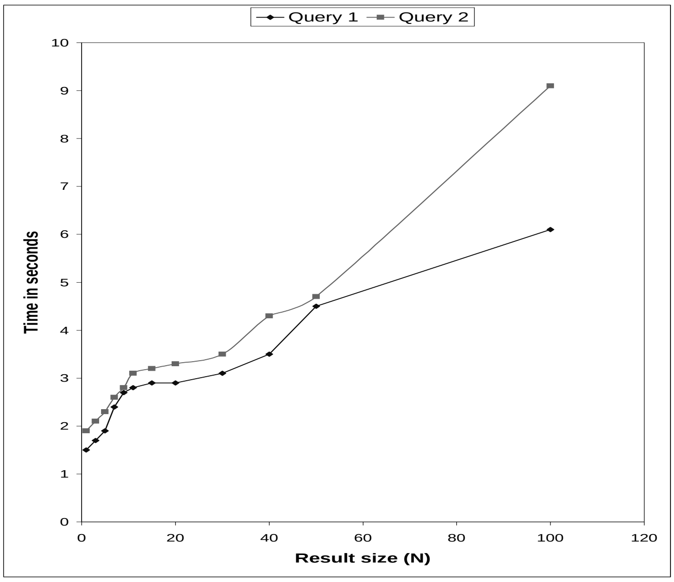

Figure 13: Total time taken as a function of result size(N)

4.2.3 Result size N

In Figure 13 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 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 iDiff 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] T. M Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley and Sons, Inc., 1991.

[DBM98] DBMS. Open olap in intelligent enterprise. *DBMS Magazine*, April 1998. http://www.dbmsmag.com/9804d14.html.

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

[FM85] P. Flajolet and G.N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31:182–209, 1985.

[Ger96] C. Gerber. Excavate your data. *Datamation*, May 1 1996. http://www.datamation.com/PlugIn/issues/1996/may1/05asoft3.html.

[Han98] J. Han. Towards on-line analytical mining: An integration of data warehousing and data mining. *DB Summit: "*http://www.dbsummit.com/articles/Han/han.html*"*, 1998.

[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.

[IDA99] Intelligent Data Analysis Group IDAG. Olap vendors increasingly see data mining integration as potent differentiator. http://www.idagroup.com/v2n0701.htm, March 1999.

[Inc96] Information Discovery Inc. Olap and datamining: Bridging the gap. http://www.datamining.com/datamine/bridge.htm, 1996.

[Mic98] Microsoft Corporation, http://www.microsoft.com/data/oledb/olap/spec/. *OLE DB for OLAP version 1.0 Specification.*, 1998.

[Sar99] S. Sarawagi. Explaining differences in multidimensional aggregates. In *Proc. of the 25th Int'l Conference on Very Large Databases (VLDB)*, 1999.

[SDNR96] A. Shukla, P.M. Deshpande, J.F. Naughton, and K. Ramasamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *Proc. of the 22nd Int'l Conference on Very Large Databases*, pages 522–531, Mumbai (Bombay), India, September 1996.

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

[SS00] S. Sarawagi and G. Sathe. $\mathbf{i^3}$: Intelligent, Interactive Investigaton of OLAP data cubes. In *Proc. ACM SIGMOD International Conf. on Management of Data (Demonstration section)*, Dallas USA, May 2000. http://www.it.iitb.ernet.in/ sunita/icube.

[Tho98] E. Thomsen. Olap and data mining: Creating a total dss solution. *DB Summit: "*http://www.dbsummit.com/articles/Thomsen/thomsen.html*"*, 1998.