# User-adaptive exploration of multidimensional data

Sunita Sarawagi

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

**Abstract**

In this paper we present a tool for enhanced exploration of OLAP data that is adaptive to a user's prior knowledge of the data. The tool continuously keeps track of the parts of the cube that a user has visited. The information in these scattered visited parts of the cube is pieced together to form a model of the user's expected values in the unvisited parts. The mathematical foundation for this modeling is provided by the classical Maximum Entropy principle. At any time, the user can query for the most surprising unvisited parts of the cube. The most surprising values are defined as those which if known to the user would bring the new expected values closest to the actual values. This process of updating the user's context based on visited parts and querying for regions to explore further continues in a loop until the user's mental model perfectly matches the actual cube. We believe and prove through experiments that such a user-in-the-loop exploration will enable much faster assimilation of all significant information in the data compared to existing manual explorations.

## 1 Introduction

We propose a new method for interactively exploring multidimensional OLAP data cubes [GCB+97] that continuously adapts to what the user knows about the data and uses that to guide him to the parts of the cube that he will find most informative. We provide a method of personalizing OLAP exploration tools so that the user for which it is trained is only shown regions that he will find surprising. Often in large corporations a single OLAP data source is deployed by users at various levels of experience with the cube. There are local store managers who are very familiar with the details of just one store; top executives who know top-level trends but none of the details and recent hire analysts who know nothing about the data but need to subsequently understand a lot of it. Currently, all these three categories of users get the same view of the OLAP data cube which they explore manually using the basic drill-down, rollup, pivot and select operator. Apart from these basic tools there is little support provided for meaningfully exploring large databases. We propose that users be allowed to navigate a data cube based on information content of the region rather than the current approach of basing it on navigational reachability or random user guesses.



**Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.**

### 1.1 Overview of the system

The system maintains a profile for each user that has an account with the OLAP system. The profile stores the parts of the cube with which he is already familiar. This profile is built either by the frontend exploration tool monitoring the amount of time the user spends with each view of the cube or by the user explicitly marking a given view of the cube as visited. The system then uses this profile to model the user's expectation about the unvisited parts of the cube. The classical maximum entropy principle is used to provide a unified way of piecing together the information in the scattered visited parts of the cube with which the user is familiar. According to this principle, the best guess expected values are those that maximize the uniformity of the data values while agreeing with all partial sums that the user has seen.

Next we define the information content of an unvisited value as the gap between the actual values and the new expected values if the user had known this value. The user can query this information in a variety of interesting ways to improve his data exploration experience. For instance, during normal exploration, starting from an initial view of the cube the user can query for the most informative path for drilling down further. Or, after having explored the cube for some time, he can ask for the ten most informative cells from anywhere in unvisited data.

As the user explores more regions of the cube, the user's profile gets enhanced and the expected value of unvisited parts is continuously updated. This process of updating the user's context based on visited parts and querying for regions to explore further continues in a loop. In each iteration the user's mental image of the cube gets closer to the actual cube until they both become one and the same. When the user stops the exploration, the context is recorded and digested by the system so that next time when the analyst logs into the cube database, the memory of what parts he has already visited is revoked from the system's log to guide further exploration.

| 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.

### 1.2 Illustration

We next illustrate the working of the system using a real-life dataset obtained from International Data Corporation (IDC). The data gives the total yearly revenue in millions of dollars for different software products from 1990 to 1994. The schema as shown in Figure 1 consists of four dimensions Product, Platform, Geography and Time and a three level hierarchy on the Product and Platform dimension.

Consider two kinds of users exploring this dataset. The first user has no prior knowledge of any part of the data and the second user has full familiarity with all the data except for the most recent two years. However, the second user has observed the total revenues for these two years.

Using the existing OLAP exploration operations (like "drill-down" and "roll-up") the first user could launch the process of understanding the data by navigating through subsets of the cube viewed at various levels of aggregation. However, the process of understand data could be long and tedious involving perhaps repeated visits to the same parts of the data. We contrast this with the experience the user would have with the new focussed exploration that this tool provides. Not knowing anything else about the user, the tool starts out modeling the expected value of each non-empty cell to be the same. The user can then query for the most informative views of the cube. The first output (shown in Figure 2) is the Platform dimension showing only ten (out of a total of 43 members) that it found most informative. The remaining are summarized by their average value in the topmost row. A second query for the next informative path returns the product dimension with eight (out of 67) distinctive members as shown in Figure 3. The year dimension shown in Figure 4 has the smallest divergence between member values and is shown last. At all time, a status bar displays how far the user's expectations are from the real values. After showing the aggregates along the four dimensions the status bar would show that 30% of the information in the data is captured. For the remaining 70% the user needs to dig deeper. Depending on the user's interest, he could either quickly ask for the top few informative cells from anywhere in the cube or follow informative paths for drilling down further. Another interesting possibility is to put the system on auto-pilot where he will be driven through the most informative views in the cube in a sequence.

The second more informed user would perhaps use the tool differently. He is already familiar with most of the detailed data except the last two years of 1993 to 1994. The tool has kept track of this fact about the user. Based on previous trends he expects an increase in revenue from 1992 to 1993 to 1994 for each Product,Platform,Geography combination. Therefore, we can directly query the tool for the most informative regions from anywhere in the cube. The results are shown in Figure 6. In the figure, the last column marked "Expected" shows the values that he would see if his extrapolations were correct and the column before it shows the actual values. These are all cases that correspond to a significant drop or increase in 1993 or 1994. For instance, based on prior knowledge the user expected the sales in 1993 for (Other Office Apps, Wester Europe and Multiuser Mainfram IBM) to be 78 whereas the actual was just 3.05. Thus, the second user will only have to concentrate on these few violators which cannot be extrapolated based on his experience of past data and the yearly totals.

### 1.3 Contents.

The rest of the paper is organized as follows. We present our formulation in terms of the maximum entropy principle in Section 2. In Section 3 we show how we enable practical implementations in large OLAP datasets and validate with experimental results. In Section 4 we present related work and finally conclusions appear in Section 5.

## 2 Formulation

Our underlying data is a $n$ dimensional cube where each cell is associated with a real value say, total sales or total number of units sold. The user sees different partial views of the data in terms of the sum of values of some subset of the cube. From this partial view he implicitly forms an expectation of the values in the cube. Our goal is to recapture these expected values.

Consider first the case where $n = 1$ and assume that there are 10 total cells along that single dimension. Suppose the user views only the sum of the 10 values in the cell. Let that sum be "1". Knowing nothing else about the data or the user's mindset, what values can we assume for each of these cells? There are an infinite number of possible 10 values that sum up to 1. One possibility is to let the values of the first cell $p_1 = 1$ and the rest of the values $p_2$ to $p_{10}$ be zero. Another is to let $p_1 = p_2 = 1/2$ and $p_3$ to $p_{10}$ be zero. Both these alternative make rather bold statements based on the limited knowledge of the data. A safer bet is to let $p_i = 1/10 = 0.1$ for all values.

Suppose if we get another view in the form of the sum of values from $p_1$ to $p_5$ to 0.75. What is the best revised guess we can make now? Following the same logic our best guess is for the first half of the values to be 0.15 and the last half 0.05. Again suppose the

| PLAT_T | PLATFORM | Act | Exp |
|---|---|---|---|
| (Each)- | (Each)- | 1.81 | 7.7 |
| Unix S. | Each) | 2.23 | 7.7 |
| Wn32 | 16-bit Windows/DOS | 24.7 | 7.7 |
| Other M. | (Each)- | 13.9 | 7.7 |
| Other M. | Multiuser Mainframe IBM | 39.2 | 7.7 |
| Other M. | Multiuser Windows NT Se | 1.23 | 7.7 |
| Other M. | Multiuser OS/2 | 1.76 | 7.7 |
| Other M. | Multiuser Other Server | 1.75 | 7.7 |
| Unix M. | Multiuser UNIX | 20.2 | 7.7 |
| Unix M. | Multiuser UNIX SCO Unix | 0.09 | 7.7 |
| Unix M. | Multiuser UNIX SGI Irix | 0.12 | 7.7 |
| Unix M. | Multiuser UNIX Other Inte | 0.06 | 7.7 |

Figure 2: Most informative drill-down dimension and its top few informative members.

| PROD_CATEC | PRODUCT | ACT | EXP |
|---|---|---|---|
| (Each)- | (Each)- | 7.16 | 7.7 |
| Vertical Apps | (Each) | 12.7 | 7.7 |
| Middleware | (Each) | 0.5 | 7.7 |
| Other | (Each) | 0.8 | 7.7 |
| System SW | (Each) | 30.6 | 7.7 |
| Info. tools | (Each) | 3.7 | 7.7 |
| Develop. tools | DBMS Engines ( | 15.7 | 7.7 |
| Develop. tools | Object-Oriented | 2.2 | 7.7 |
| Develop. tools | Object CASE | 0.9 | 7.7 |

Figure 3: Second most informative dimension.

| Year | ACT | EXP |
|---|---|---|
| 1990 | 6.54 | 7.7 |
| 1991 | 6.75 | 7.7 |
| 1992 | 8.39 | 7.7 |
| 1993 | 9.58 | 7.7 |
| 1994 | 7.23 | 7.7 |

Figure 4: Least informative of the four dimensions.

Figure 5: Information content of various dimensions. The last column denotes expected value. The "ACT" column represents actual values.

| PRODUCT | GEOGRAPHY | PLATFORM | YEAR | ACTUA | EXPEC |
|---|---|---|---|---|---|
| Other Office Apps | Western Europe | Multiuser Mainframe IBM | 1993 | 3.05 | 78.25 |
| EDA | Western Europe | Multiuser UNIX | 1994 | 19.97 | 142.41 |
| Operating Systems | United States | Multiuser Other Server | 1994 | 0.22 | 47.82 |
| Operating Systems | Western Europe | Multiuser Minicomputer O | 1993 | 3.27 | 69.74 |
| Middleware | Asia/Pacific | Multiuser Mainframe IBM | 1993 | 185.72 | 96.10 |
| CASE (Non-Object) | United States | 16-bit Windows/DOS | 1994 | 1.77 | 57.58 |
| DBMS Engines (Non-Object) | United States | Multiuser Mainframe IBM | 1994 | 90.31 | 315.03 |
| DBMS Engines (Non-Object) | Rest of World | Multiuser Mainframe IBM | 1994 | 11.00 | 98.92 |
| 4GL & Report Writers | Rest of World | Multiuser Mainframe IBM | 1993 | 0.31 | 42.48 |
| 3GLs & Develop. Environments | Rest of World | Multiuser Mainframe IBM | 1993 | 0.62 | 38.67 |

Figure 6: Informative regions returned to user who is familiar with the entire cube for years 1990-1992 but unfamiliar with years 1993 and 1994.

user sees a third view of the data consisting of sums of values from $p_3$ and $p_7$ and let that sum be 0.5. In this case, it is not all that obvious how we distribute the three partial sums to derive individual values. Fortunately, this is a classical problem with links to biblical times that has found widely accepted answers in the Maximum Entropy principle [BPP96, GS85].

The maximum entropy principle states that given a collection of facts choose a model that is consistent with all the facts but otherwise is as uniform as possible. A mathematical measure of the uniformity of a distribution is provided by entropy defined as $H(p) = -\sum_{i=1}^{m} p_i \log p_i$, where $p_i$ denotes the estimated value or probability of the $i$th cell. The entropy is bounded from below by zero, the entropy of a model with no uncertainty at all i.e., $p_i$ is either 0 or 1 for all $i$. It is bounded from above by $\log m$, the entropy of the uniform distribution where all $p_i$ have the same value of $\frac{1}{m}$. Our goal is to choose the distribution $p$ that maximizes $H(p)$ while satisfying the constraints imposed by the partial visited views of the data. A constraint $C_i$ is a restriction on some subset of these $m$ values to sum up to some observed value $\tilde{\mathrm{p}}(C_i)$. In the example above we had three such constraints with $\tilde{\mathrm{p}}(C_1) = 1$, $\tilde{\mathrm{p}}(C_2) = 0.75$ and $\tilde{\mathrm{p}}(C_3) = 0.5$. The final optimization problem is:

$$\begin{aligned} \max_p H(p) &= \max_p(-\textstyle\sum_{i=1}^{m} p_i \log p_i) \quad \text{such that} \\ \forall C_i, \textstyle\sum_j p_j I_{ij} &= \tilde{\mathrm{p}}(C_i), \\ \text{where} \quad I_{ij} &= \begin{cases} 1, & \text{if cell } j \text{ is included in } C_i, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

This optimization problem is the mathematical essence of the Maximum Entropy philosophy that according to E.T. Jaynes [Jay90] *"agrees with everything that is known, but carefully avoids assuming anything that is not known"*.

### 2.1 Finding the best values of $p$

The objective function $H(p)$ always has a unique solution as long as the constraints are consistent [PPL97]. In most cases finding that unique solution through any closed form formula is not possible. However, there are well defined iterative algorithms that are based on the observation that the optimal $p$ values can be expressed in the following product form.

$$p_j^{\mu} = \mu_0 \prod_{C_i} \mu_i^{I_{ij}} \tag{1}$$

We use $p^{\mu}$ to denote the class of $p$ values that can be expressed in the above product form and $p_j^{\mu}$ is the expected value of the $j$ cell. For each constraint $C_i$

there is a term $\mu_i$. The term $\mu_0$ is a normalization constant to ensure that the probabilities sum up to 1.

#### 2.1.1 The Iterative scaling algorithm for finding best $p$

Start with $\mu_i = 1$ for all constraints.
Update $\mu_0$ so probabilities sum to 1
While the $\mu_i$s have not converged
  For each constraint $C_i$
    Update $\mu_i$ by scaling with $\tilde{\mathrm{p}}(C_i)/p(C_i)$
    Recalculate expected values $p$ using Equation 1
Update $\mu_0$ so probabilities sum to 1

The above algorithm is guaranteed to converge to the optimal solution as long as all constraints are consistent [PPL97].

### 2.2 Finding informative constraints

The second part of our problem is to find the most informative constraints from unvisited data. We define such a constraint to be the one that reduces the distance between the actual values $\tilde{\mathrm{p}}$ and expected values $p^\mu$ by the maximum amount. We measure distance using the traditional Kullback-Leibler divergence criteria defined as:

$$D(\tilde{\mathrm{p}}||p^\mu) = \sum_j \tilde{\mathrm{p}}_j \log \frac{\tilde{\mathrm{p}}_j}{{p^\mu}_j}$$

Let $p^C$ denote the expected values after the addition of the first $C$ constraints and let $p^{C+f}$ denote the expected values after adding a new constraint $f$. Our goal is to pick the $f$ that reduces the distance by the maximum amount, i.e.,

$$f = \mathrm{argmax}_f(D(\tilde{\mathrm{p}}||p^C) - D(\tilde{\mathrm{p}}||p^{C+f})) \quad (2)$$
$$= \mathrm{argmax}_f \textstyle\sum_j \tilde{\mathrm{p}}_j(\log p_j^{c+1} - \log p_j^c). \quad (3)$$

From equation 1 we can write $p^C$ as a product of $|C|$ terms one corresponding to each constraint $c_j \in C$. The values of coefficients $\mu_j$ could change due to the new constraint $f$ but for reasons of efficiency we ignore this change and only take into account the change with the addition of the new coefficient $\mu_{c+1}$. Based on this assumption we can write Equation 2 using results from Equation 1 as:

$$f = \mathrm{argmax}_f \textstyle\sum_j \tilde{\mathrm{p}}_j \log \mu_{c+1}^{I_{(c+1)j}} \quad (4)$$
$$= \mathrm{argmax}_f \textstyle\sum_j \tilde{\mathrm{p}}_j I_{(c+1)j} \log \frac{\tilde{\mathrm{p}}(f)}{p^c(f)} \quad (5)$$

## 3 Adapting the maximum entropy principle to OLAP data

The main challenge in adapting the maximum entropy principle to OLAP data is handling the scale. Traditional applications have concentrated on small datasets and therefore there is little previous literature on scaling the iterative algorithm and the search for new constraints, both these are computationally expensive procedures. Also our goal is to be able to interactively furnish the next few informative constraints even while the user's context is continually being changed as he navigates around the data. We next discuss a collection of optimizations that we applied on these methods to make them efficient on large OLAP datasets. We also present empirical evidence of their usefulness through experiments on several OLAP datasets. In Section 3.1 we present details of the experimental setup. In section 3.2 we present a number of optimizations for improving the first part of our tool, that is, updating the expected values with the addition of new constraints. In section 3.3 we present optimizations for getting answers to finding the most informative regions. Finally, in Section 3.4 we discuss issues in integrating this tool with a OLAP system.

### 3.1 Experimental setup

We used the following datasets for our experiments.

**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.

**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. The numbers within bracket denote the cardinality of that level.

| 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) | | | |

**Student data:** This data is about the enrollment statistics of a university with dimensions as shown in the table below. The total number of cells at detailed level is 4560 which is very small by OLAP standards. We therefore do not use this dataset for performance studies. However, it is useful for doing a qualitative assessment of our method because the dataset is real.

| Student | Sex | Program | Department | Year |
|---|---|---|---|---|
| Category (9) | Sex (2) | Name (10) | Name (28) | Year (10) |
| | | Category (3) | | |

**Grocery sales data:** This is a demo dataset obtained from the Microsoft DSS product [Mic98]. 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) | | |

These experiments were done on a PC with a 333 MHz Intel processor, 128 MB of memory and running Windows NT 4.0. A DB2 ROLAP database was used to process the queries.

**Workload** We simulate a user's exploration of the data cube using the following model. The user starts at the topmost level where all dimensions are aggregated to a single value. At any time, the user views data in the context of at most two dimensions at a time. Remaining dimensions are either aggregated or selected on a single value at any level of its hierarchy. From one view of the cube the user moves to a neighboring view as follows: Select one dimension $d_i$ from the two dimensions that are currently either row or column and select another dimension $d_j$ from the remaining set to replace $d_i$. Fix the value of $d_i$ to either one of its members or aggregate it to level "All". This yields a new view of the cube from which the user can move to a neighboring view using the same procedure.

**Notation** We introduce some notations. Consider a cube with four dimensions $A, B, C$ and $D$. We use the term *view* to denote the different parts of this cube that the user has visited. A view represents a collection of constraints. For instance, if the user has viewed the totals along dimension $A$, then the view is said to be $A$ and this view consists of as many constraints as there are members along $A$. The lower case letter $a_i$ denotes the $i$th member of dimension $A$. If the next view the user visits is $a_iB$ i.e., for fixed value $a_i$ of dimension $A$ he is viewing totals along each member of dimension $B$, then the constraints that they represent have the form $a_ib_j$ where $b_j$ spans over all the members of dimension $B$. We will sometimes call a view a constraint where the distinction is not important.

### 3.2 Optimizing the expected value update process

We first present ways of optimizing updates to the expected values of the detailed cube with the addition of a new constraint. Our requirements of long-term memory of the users' context requires an incremental formulation where a persistent storage is used to keep track of the constraints and the partially computed expected values. We maintain two pieces of information for each <cube,user> pair. First is a list of the set of constraint imposed by the user and second is expected values based on the context established so far.

#### 3.2.1 Optimized representation

The first optimization relates to how we store the currently computed expected values. Instead of keeping a separate entry for each detailed cell as implied by the iterative algorithm we group together and keep a single entry for each *contiguous* region that will have the same expected value with the current set of constraints. For instance, if the only constraint that we have is $A$ then all detailed cells with same value of dimension $A$ will have the same expected value, hence we store only as many entries as the domain of $A$ instead of storing the expected value for each detailed cell in $ABCD$. When the user submits a second constraint $D$ all cells with the same value of dimensions $A$ and $D$ will have the same expected value, hence we store the expected values at the $AD$ level.

This optimized representation makes the addition of a new constraint more complicated. Every time a new constraint $f$ is added, we might need to De-aggregate the level to which a region is stored. For instance, in our previous example adding a third constraint $a_iCd_j$ would require us to expand the entry $a_id_j$ with the new dimension $C$. From these new regions we remove any region already materialized. More details of this step appear in the expanded version [Sar00a].

The optimized representation not only reduces storage requirements but also improves the iterative algorithm because the iterations are performed over aggregated values.

**Improvements achieved on experimental datasets** We demonstrate the impact of this optimization by measuring the speed up obtained by the iterative procedure for the datasets and query workload discussed earlier. In the graphs in Figure 10, the $X$ axis represents the constraint in the order in which they are submitted to the system and the $Y$ axis denotes the total time for the constraint propagation. We plot two graphs, one for the optimized representation (marked "opt") and second for the detailed representation where the expected values are at the detailed level (marked "noopt"). From the graphs, we observe a factor of five improvement in total time for the software data and even greater (between factors of 10 and 100) reduction with the larger datasets. This difference is significant because it helps cross the boundary between interactive and batch processing. For the larger datasets, operations that previously required 10 minutes can be completed in half a minute making interactive sessions more feasible.

#### 3.2.2 Optimizing iterative process

We next present optimizations for reducing the number of iterations needed for convergence and also pruning the number of constraints involved in each iteration.

We first introduce some definitions for formalizing the relationships between constraints. A constraint $C_i$ is said to *subsume* another constraint $C_j$ if the sum at $C_i$ includes all elements that are included in $C_j$. Thus,

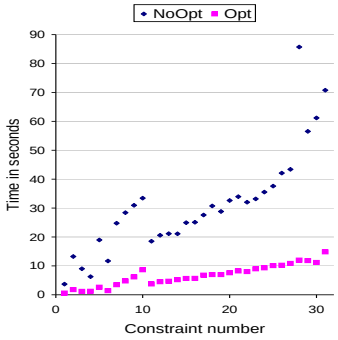


Figure 7: Software revenue data.

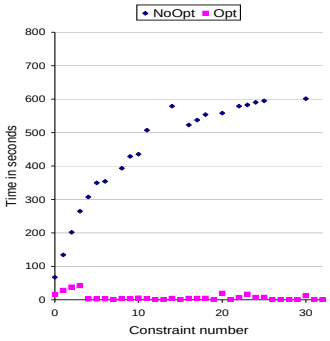


Figure 8: Grocery sales data.

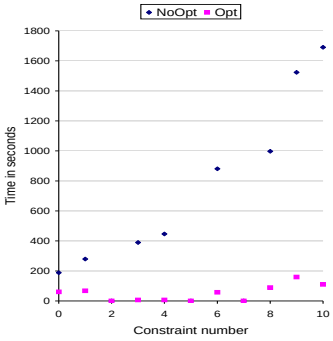


Figure 9: Olap benchmark.

Figure 10: Improvement due to optimized representation of expected values. 'Opt' and 'NoOpt' denote total time with and without the optimizations. $Y$ axis is total time in seconds and $X$ axis the number of constraints submitted.

constraint $a_i$ subsumes constraint $a_i c_k$. A view $V_i$ is said to be more detailed than another constraint $C_j$ if $V_i$ aggregates the same set of values as $C_j$ but $V_i$ includes more than one constraint. Thus, view $a_i B$ is *more detailed* than constraint $a_i$ because together they cover all cells where dimension $A$ has value $a_i$ but view $a_i B$ has separate constraints corresponding to different values of $B$.

We exploit these relationships to speed up the iterative procedure when a new constraint is added.

**Minimize overlap between constraints** First, when we get a new view say $a_i B c_k$, we find all existing constraints that subsume it (example $a_i$) and exclude from each of them the subsumed part. For example, an existing constraint, $a_i$ would be replaced by a modified constraint $a_i \overline{c_k}$ that excludes any cell where dimension $C$ has value $c_k$. Sometimes this might cause an existing constraint like $a_i c_k$ to be eliminated totally. Similarly, from the new view we exclude the constraints that are subsumed by it. For example if there is a constraint $a_i B c_k d_l$ then we modify the new constraint $a_i B c_k$ to be $a_i B c_k \overline{d_l}$.

Rewriting thus significantly reduces the number of iterations because of the reduction in the overlap between constraints. In the modified form $a_i B c_k$ and $a_i \overline{c_k}$ have no cells in common. Therefore, only *one* iteration is needed for convergence with these two constraints.

**Prune constraints** The algorithm of Figure 2.1.1 cycles through *every* constraint in an iteration. We suggest pruning from the current iteration those constraints whose estimated impact on the expected values is small. When a new constraint is added, we apply it first. Subsequently we apply only those constraints whose estimated change is greater than a small threshold. Clearly, if a constraint has no overlap with any of the constraints before it, it can be safely pruned from the current iteration. For others, we estimate expected change as follows: For each constraint applied before it in the current iteration we know the maximum change of any expected values due to this constraint. Let $\delta_j$ denote this maximum change on a cell due to a constraint $C_j$. For a constraint $C_k$ at the $k$th position of the current order of constraints, we calculate the estimated maximum change per cell $\hat{\delta_k}$ as

$$\hat{\delta_k} = \sum_{i=1}^{k-1} \text{influence}(C_i, C_k)\delta_i.$$

and skip those constraints for which this estimated maximum change is smaller than a threshold. We quantify the influence$(C_i, C_k)$ of a constraint $C_i$ on another constraint $C_k$ by the fraction of the aggregated values of $C_k$ that overlap with $C_i$. For instance, if $C_i$ is $a_i$ and $C_j$ is $d_k$ and there are 100 cells with $D$th dimension member $d_k$ and 10 of them have dimension $A = a_i$ then the influence of $C_i$ on $C_k$ is $10/100 = 0.1$. If there is a third constraint $C_l = b_j$ that sums up 1000 entries and 20 of them overlap with $C_i$, then influence of $C_i$ on $C_l$ will be $20/1000 = 0.02$.

**Improvements achieved on experimental datasets** We demonstrate the impact of these optimizations on the iterative algorithm. The setup and the axes are the same as in Section 3.2.1. Data is assumed to be stored in the optimized representation of Section 3.2.1. In Figure 14 we show two plots for each datasets. One plot is for the optimized iterative algorithm (marked "order") and the second without these optimizations (marked "opt"). We notice from the graphs that these optimizations give us another around a factor of two reduction in total time. In the initial stages when the number of constraints is small, the improvement is lower as expected and it increases as more constraints get added.

### 3.2.3 Asynchronous batched computation

A third optimization we propose is batching updates due to multiple constraints. When the user submits a view, the request is queued and the user call returned.

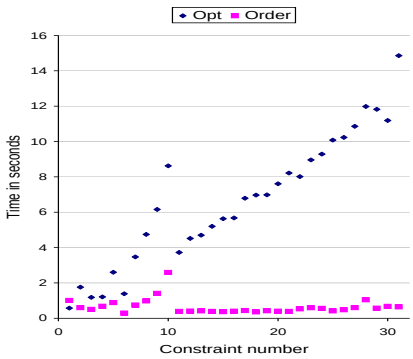


Figure 11: Software revenue data.


Figure 12: Grocery sales data.

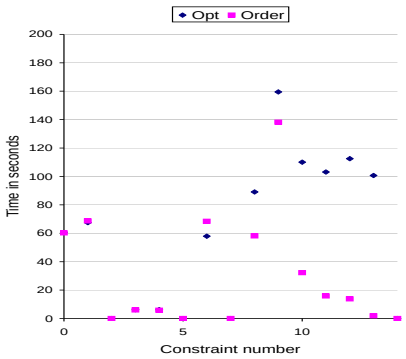


Figure 13: Olap benchmark.

Figure 14: Improvement due to removing subsumed constraints and ordering and pruning constraints. 'Order' and 'Opt' denote total time with and without these optimizations respectively. $Y$ axis is total time in seconds and $X$ axis the number of constraints submitted.

The user does not wait for the effect of the new constraints to be propagated. Thus registering a view as visited is instantaneous. A separate thread is used to asynchronously refine the expected values via the iterative process. An offshoot of this architecture is that updates due to multiple constraints can be batched and also redundant constraints removed. For instance, if the user submits a view $A$ followed soon after by another view $AB$ the first view would be removed as redundant. We batch iterations due to multiple constraints as follows. We do not invoke a new round of iterative improvements every single time a constraint is added. Instead, as long as there are new constraints in the queue we apply just that constraint to update the expected value. When no more constraints new constraints are waiting we invoke the iterative algorithm to refine the expected values.

### 3.3 Optimizing the constraint selection process

Our goal is to use the expected values found in the previous step and the original data cube to report the most informative regions in the data cube. The query for information regions can be posed in a number of different ways as discussed in Section 1.1. These queries can be classified into two broad categories. One class requires the most informative contiguous region starting from some initial view of the cube. The second class requires for the top few informative constraints from anywhere in the cube. We expect the first class of queries to be more frequent at the top levels of the cube when the user is relatively unfamiliar with the rest of the cube. The second class of queries are more likely when a user is familiar with most of the cube and just needs to search for interesting information in detailed data that he might have missed. In terms of computation load, the first class of queries are easier to compute because the user has significantly reduced the portion of the cube to be searched through his starting context. The second class of queries are more challenging since they require searching the entire cube and also because the constraints could interact with each other in arbitrary ways. Including a constraint of the form $a_i b_j$ changes the information content of constraint $a_i b_j c_k$ and viceversa. Such interactions are not present in the first class of queries because the constraints are all from the same view of the cube and thus cover non-overlapping data. We concentrate on the second class of queries since the first type are straightforward.

The user supplies a parameter $N$ that denotes the maximum number of constraints he is interested in inspecting. We need to return the set of $N$ constraints that are most informative. Using Equation 2 we define the information content of our final set of $N$ chosen constraint as the increase in likelihood due to the new expected values after all the $N$ constraints have been applied to the data. This global objective function is hard to evaluate. When $N = 1$, that is, when we want the single most informative constraint we can simplify Equation 2 to Equation 4 which quantifies the informative content of a constraint as $\sum_j \tilde{\mathrm{p}}_j I_{(c+1)j} \log \frac{\tilde{\mathrm{p}}(f)}{p^c(f)}$ i.e., the sum over the actual values of all cells included in the constraint multiplied by a scaling factor that is the same for all the cells. One option is therefore to find the most informative constraint first, incorporate its effect on the data, find the next most informative constraint and so on upto $N$ constraints. Not only is this solution computationally expensive, it also does not guarantee optimality. We need a method that finds the $N$ constraints simultaneously and ideally in one pass of the data. The main difficulty is that unlike for the case of $N = 1$ the new expected values $p_{c+1}$ are hard to evaluate in closed form when there are multiple constraints in the final answer affecting it. To enable practical solution, we restrict the class of $N$ constraints to be those that either totally subsume each other

or do not overlap at all. We then use the Remove-Subsumed optimization of the previous section to remove from each constraint the part subsumed by some other more detailed subset. Consequently all cells in the cube are now covered by at most one of the $N$ constraint. Even with this restriction finding the optimal solution is non-trivial because of the interactions between subsumed constraints. Including a constraint of the form $a_i b_j$ changes the worth of including a second constraint of the from $a_i b_j c_k$ and viceversa.

In [Sar99] we faced a similar challenge when attempting to find the best $N$ row summary of the difference between two subcubes. We solved the problem by developing an efficient one-pass dynamic programming algorithm that is close to the optimal answer in certain special cases. We directly apply that algorithm. The algorithm starts with a bottom-up scan of the most detailed data and then aggregates tuples to higher levels while at the same time constructing the best solution. More details of the algorithm appear in [Sar99].

### 3.3.1 Experimental results

We present experimental evaluation of the overall system after including all the optimizations suggested. We evaluate our system along two important metrics: performance and quality of data exploration.

**Timing measurements** First we show overall response time to the top-$N$ informative feature to demonstrate feasibility in a practical setting. The user interacts with the system in two ways: first by registering part of the constraints as seen and second by querying for informative regions. The response time for the first part is instantaneous because of asynchronous processing. The main concern is about response time of the second part. However, before responding to these queries, we need to ensure that processing on all constraints submitted prior to it has been completed.

For response time measurements we augment the workload in Section 3.1 with timing information. The time spent on one view of the cube is set to be a function of the number of cells in the current view. We assume that per cell the user spends an average of one second distributed randomly from 0 to 2 seconds. Thus, a view with 20 cells would be stared at for twenty seconds before the user navigates to the next neighboring view. Periodically the user queries for the ten most surprising constraints given his current view of the cube. We assume the periodicity to be distributed randomly between one and ten navigation of the cube.

Figure 18 shows the response time for the top-$N$ informative constraints as a function of the number of constraints after which the query is posed. We find that even for the largest data which is the OLAP benchmark with 1.3 million tuples we are able to return the best answer within 2 minutes that makes it possible to deploy our tool in an interactive setting. Our experiments were run on very modest hardware. More powerful servers keeping pace with Moore's bounty can reduce the response time even further. We report performance results on other OLAP databases in the expanded version of this paper [Sar00a].

**Exploration quality** We measure the rate of information transfer to the user with our new focussed search. For this we measure the gap between the actual and expected values under two scenarios. In the first case, the user marks as visited the constraint that the system returns as the most informative. In the second case, we simulate a random exploration similar to the workload described in Section 3.1. In Figure 22 we plot the relative square error between the actual and expected values against the number of constraints. As expected, in both cases as more and more constraints get added the error reduces. For the student data we notice a remarkable reduction in error where it reduces from 0.9 to 0.18 within just 50 constraints (1% of total data size). In other words 1% of the data captures more than 80% of the information content. Similarly, we notice that for the Software revenue dataset error reduces to 0.73 within 50 constraints (0.2% of total data size). That is, just 0.2% of the total data size can explain 25% of the information content. The Grocery sales data is a synthetic datasets – therefore we notice that they start out with very low information content. Just by including the total sum at the topmost level, we explain 70% of the information in the data. After that we get little improvement with new constraints because of the high randomness in the data.

## 3.4 Integration with existing OLAP systems

Our goal is to allow persistent storage of the user's context and also to allow immediate refresh of expected values as new constraints get added. Both these requirements, make the underlying OLAP data source a natural choice for storing our intermediate results. For each <cube,user> pair we maintain an Expected-cube that stores the expected values at various aggregate levels using the optimized representation of section 3.2.1. Our tool resides as an attachment to the OLAP system that collects the user interactions and handles all optimization logic. All data intensive tasks are pushed to the OLAP server through dynamically generated queries. For instance, when a new constraint is submitted we need to adjust the expected values using the formulas in 2. This requires first aggregating the Expected-cube up to the level of the constraint to get the scale factor as the ratio of the observed and expected value for each member of constraint. Next we update the Expected-cube by multiplying with the scale factor using a join. Our prototype works on IBM's DB2/UDB's ROLAP features (version 6.1)

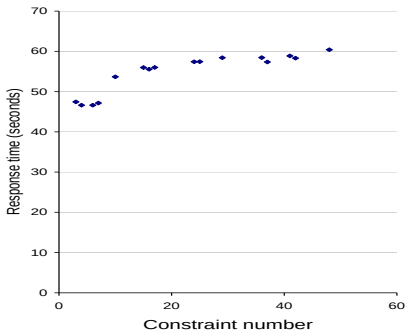


Figure 15: Software revenue data.

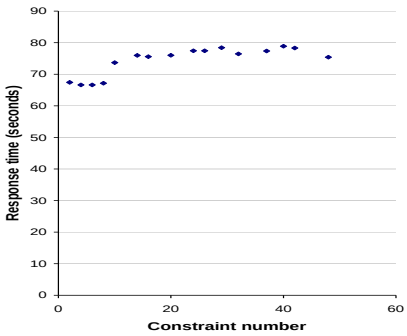


Figure 16: Grocery sales data.

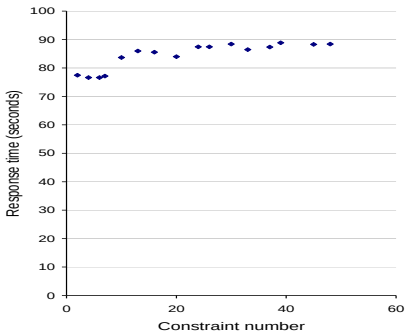


Figure 17: OLAP benchmark.

Figure 18: Response time for the $N$ most informative constraints queries. $X$ axis is the number of constraints after which query was posed and $Y$ axis is response time.

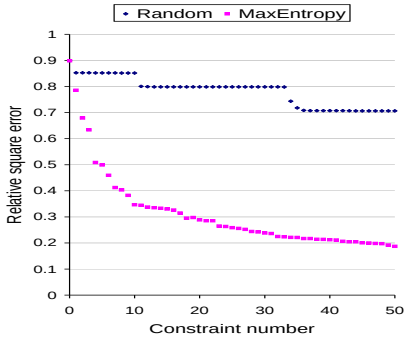


Figure 19: Student data.

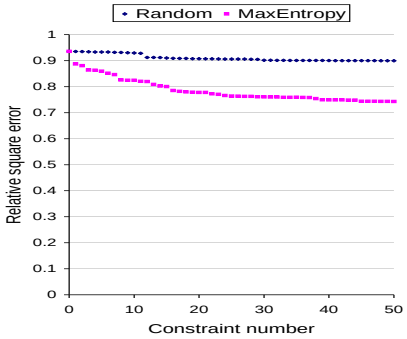


Figure 20: Software revenue data.

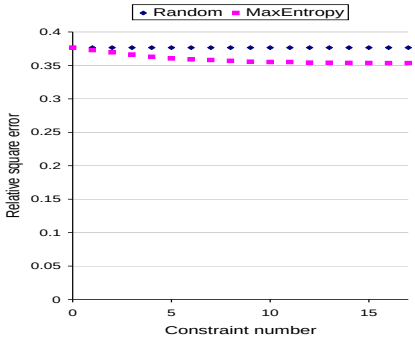


Figure 21: Grocery sales data.

Figure 22: Change in error as constraints get added. The curve 'Random' shows error for a random set of constraints and the curve 'MaxEntropy' shows error with the most informative constraints

system and Oracle's 8i system – both of which provide advanced indexing and materialized aggregate views for efficient processing of OLAP queries.

## 4 Related work

The work reported here is part of our continuing **i$^3$**project [Sar00b] on taking OLAP to the next stage of interactive analysis where we automate much of the manual effort spent in analysis. Recently, some attempts have been made to enhance OLAP products with mining primitives like decision tree classifiers [Dis, Cor97], clustering [Sof] and association rules [HF95]. In all these cases, the approach is to take existing mining algorithms and integrate them within OLAP products. The approach in the **i$^3$**project is to first investigate how and why analysts currently explore the data cube and next automate them using new or previously known operators.

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 in detailed data. We reported as exceptional any value that was significantly different from any value calculated assuming all of its subsets are known. This method has several differences with our current method of defining information content of a cell. First, the previous method computed exceptions in a batch mode whereas the current setting is online. Consequently, the interest value of a cell was derived assuming all its parents are known whereas in this project we assume only the visited parents are known. Second, the previous method used an intrinsic notion of the information content of a cell by making it a function of its own difference from the expected value. In contrast, in this case we have a more global notion where information content of a cell is measured in terms of how much knowing it bridges the gap between the expected and actual values of the entire cube. Often both definition of interestingness might return the same value but there are important cases where they differ. For instance if an aggregate value $v$ differs significantly from its expected value but otherwise the detailed values underneath it are highly divergent, then by the intrinsic criteria $v$ might qualify as interesting but it will not be so by the extrinsic criteria.

In [Sar99] we automate another area where analysts spend significant manual effort exploring the data: namely, finding reasons to explain why a certain

aggregated quantity is lower or higher in one cell $v_a$ compared to another cell $v_b$. We formulated this as reporting summarized differences between the two isomorphic cubes $C_A$ and $C_B$ that are aggregated to form the observed sums at $v_a$ and $v_b$. This summarization has close ties with the second part of our tool where we report the top $N$ informative cells from unvisited cube. The expected values cube be thought of as cube $C_A$ and the actual values of cube $C_B$ and we need to report the $N$ constraints that will best summarize the difference. In Section 3.3 we discussed how we used these results.

Another body of related work arises from our use of the Maximum Entropy principle for calculating expected values. This is a classical topic with broad based applications in several areas including physics and chemistry in the pre-computer era and more recent applications in several problems on statistical estimation and pattern recognition. A recent nice tutorial and a computer science application is presented in [BPP96] where maximum entropy is used in natural language processing to model word usage based on prior words used in a passage. In data mining [MPS99] presents a more focussed application of Maximum Entropy to the problem of frequent itemset mining.

## 5 Conclusion

In this paper we proposed a new method of interactively exploring multidimensional data cubes that guides a user on what is informative after continuously factoring for what the user has already explored. There were two key components of this tool. First, modeling a user's expectation of values in unvisited parts based on what he already knows about the data. Second, attaching a measure of information content to each unvisited part of the cube. We found a unified answer to both these issues in the time-tested philosophy of Maximum Entropy. However, multidimensional data of the scale commonly present in typical OLAP systems are not directly amenable to the expensive iterative procedures required for solving the constrained optimization problem that arises out of the maximum entropy principle. We developed a number of optimizations to make these procedures efficient. Our optimizations lead to one to two orders of magnitude improvement in total time on large OLAP datasets. Another set of experiments on real-life data showed that a guided search can significantly accelerate the understanding of the data — for one dataset just a small 3% of the data captured 80% of the information content in the entire cube. We have implemented a prototype that integrates with existing OLAP systems and capitalizes on their processing power by pushing expensive computations to the OLAP server.

The most compelling future work is providing good visualization of the entire system to visually represent the information content of the various parts of the cube and show it in the context of the user's prior knowledge. Other topics include deleting or fading away constraints and allowing user defined constraints like expected seasonality in sales values.

## References


[BPP96] A. Berger, S. Della Pietra, and V. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996. `http://www.cs.cmu.edu/afs/cs/user/aberger/www/html/tutorial/tutorial.ht\%ml`.

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

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

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

[GCB+97] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[GS85] S. Guiasu and A. Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1), 1985.

[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.

[Jay90] E.T. Jaynes. Notes on present status and future prospects. In W.T. Grandy and L.H. Schick, editors, *Maximum Entropy and Bayesian Methods*. Kluwer, 1990.

[Mic98] Microsoft corporation. *Microsoft decision support services version 1.0*, 1998.

[MPS99] Heikki Mannila, Dmitry Pavlov, and Padhraic Smyth. Prediction with local patterns using cross-entropy. In *Proceedings Knowledge discovery in databases*, pages 357–361, 1999.

[PPL97] S. Pietra, V. Pietra, and J. Lafferty. Inducing features of random fields. *In IEEE Transactions on Pattern Analysis and Machine Intelligene*, 19(4):380–393, 1997.

[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`.

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

[Sar00a] S. Sarawagi. User adaptive exploration of olap data cubes. Submission to the VLDB journal: `http://www.it.iitb.ernet.in/~sunita`, 2000.

[Sar00b] Sunita Sarawagi. **i³**: Intelligent, Interactive Investigaton of OLAP data cubes. In *Proc. ACM SIGMOD International Conf. on Management of Data (Demonstration section)*, Dallas USA, May 2000.

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