VLDB Journal manuscript No.
(will be inserted by the editor)

User-cognizant multidimensional analysis*

Sunita Sarawagi[1]

Indian Institute of Technology Bombay, Indiae-mail: sunita@it.iitb.ac.in

Abstract Our goal is to enhance multidimensional database systems with a suite of advanced operators to automate data analysis tasks that are currently handled through manual exploration. In this paper we present a key component of our system that characterizes the information content of a cell based on a user's prior familiarity with the cube and provides a context-sensitive exploration of the cube. There are three main modules of this component. A Tracker, that continuously tracks the parts of the cube that a user has visited. A Modeler, that pieces together the information in the visited parts to model the user's expected values in the unvisited parts. An Informer, that processes user's queries about the most informative unvisited parts of the cube.

The mathematical basis for the expected value modeling is provided by the classical Maximum Entropy principle. Accordingly, the expected values

* This is an expanded version of the conference paper in [Sar00]

are computed so as to agree with every value that is already visited while reducing assumptions about unvisited values to the minimum by maximizing their entropy. The most informative values are defined as those that bring the new expected values closest to the actual values. 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

Multidimensional databases form the basis of Online Analytical Processing (OLAP) [Cod93, CD97] products [Arb, Sof]. Typically categorical attributes like Products and Stores form *dimensions* and numeric attributes like Sales and Revenue form 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. A user typically navigates various subsets of the multidimensional cube using a sequence of operations like select, drill-down, roll-up and pivot conforming to the multidimensional view of data. Apart from these basic tools there is little support provided for efficiently exploring large databases.

Our goal is to take these OLAP tools to the next stage of interactive analysis where we automate much of the manual effort spent in analysis.

Towards this end, our ongoing $\mathbf{i^3}$ [SS00] project provides a suite of operators that are designed to return in a single step, and interactively, the answers to questions that users currently find through long multi-operation sessions. In [Sar99] we presented one such operator that can explain in one step why a certain aggregated quantity is lower or higher than another. In this paper we present another component of $\mathbf{i^3}$ that is responsible for efficiently locating the most informative parts of the cube. We propose a new paradigm of exploring multidimensional data that actively participates with the user to capture what a user already knows about the data and based on that dynamically updates the information content of unseen regions. This makes it possible to navigate a data cube based on information content of a region rather than navigational reachability or random user guesses.

This provides an example of personalization to the core — a goal of several software systems today. Often in large corporations a single OLAP data source serves multiple users at varying 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, roll-up, pivot and select operator. We propose that users should be able to navigate a data cube based on their prior familiarity with data. As the user gets more and more familiar with the data, the surprise-value of different parts of the data changes dynamically. The tool is cognizant of this and provides a variety of ways (elaborated in Section 2) of reaching to the current most informative regions.

The two key components of the above process are: modeling user's expectation of the unvisited parts based on arbitrary visited parts of the cube and defining what is most informative in the unvisited parts. We use the Maximum Entropy principle to solve these problems. In Section 3 we present this procedure. In Section 4 we discuss how these methods can be applied to large multidimensional OLAP databases to keep the response time within the limits of practical interactive systems. We develop procedures for *incrementally* and efficiently updating the interesting frontiers to explore further as new areas of the cube are visited. Our tool has been designed to work as a toolkit attached to any existing OLAP databases. All persistent data is stored in the OLAP engine and all heavy-weight processing is pushed to the OLAP server through dynamically generated queries.

We report results from real-life datasets to prove usefulness and feasibility in Section 5. In one real-life dataset we found that just 1% of the total cells in the cube was enough to convey 90% of the total information in the cube. In another case 0.25% of the data could capture 20% of the information. Thus by guiding the user to these informative areas of the cube we could significantly reduce the time a user has to spend in familiarizing himself with the data.

This proposed paradigm has broader application on the field of exploratory data mining. One problem that plagues the field is finding patterns that the user will find novel. This user-in-the-loop paradigm of exploration enables each user to set up a context in which only facts found interesting, after accounting for what the user already knows, will be reported. After the user inspects these results, they will be added to the user's context. Next time that the user logs on, this context will be invoked when reporting further interesting results.

2 Overview of the system

There are three main components of this system.

The first component **Tracker** is responsible for defining a user's context based on the part of the cube he is familiar with. This can be done either transparently 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 context of a user can be either persistent or ad hoc. In the persistent mode an account is created for each user where his context is maintained across login sessions. In the ad hoc mode, the user can define a small context for one-time querying of a portion of the cube. This context is forgotten after the ad hoc query.

The second component **Modeler** calculates expected values of unvisited parts of the cube based on the user's context. The user can limit the part of the unvisited cube on which expected values will be computed. The classical maximum entropy principle is used to provide a unified way of extrapolating from the information in the visited parts of the cube. According to this principle, the best guess expected values are those that maximizes the likelihood of the actual values while agreeing with all partial sums that the user has seen and making the least additional assumptions about the data. Exact mathematical details of this step are deferred to Section 3.

The third component **Informer** is used to query for the informative cells in the unvisited parts of the cube. We define the information content of an unvisited value as the gap between the actual values and the new expected values if this value had been part of the user's context. The user can query this information in a variety of interesting ways, in batch or interactive modes, to improve his data exploration experience. The queries are integrated with the existing data explorations tools in an unobtrusive manner — the user can continue to explore the data as usual and only when he needs to be guided to the informative regions need the Informer be involved. We present some example scenarios to show some interesting possibilities.

The three components interact in a loop. As the user explores more regions of the cube, the Tracker updates the user profile. The Modeler then updates the expected value of unvisited parts and that in turn changes the information content of each unvisited cell. 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 become one and the same. When the user stops the exploration, the context is stored by the system so that next time when he logs on, the memory of what parts he has already visited is used to guide further exploration.

We next present a few example scenarios to illustrate the working of the system.

2.1 Scenario 1: Informative drill-down paths

Consider the following real-life dataset about the enrollment statistics of a university with dimensions as shown in the table below. The data contains the total number of registered students in each department for each year in the last decade divided into the degree program and the category and gender of the student. The values within brackets are the number of distinct members at that level of the hierachy.

Student	Gender	Program	Department	Year
Category (9)	Gender (2)	Program (10)	Dept (28)	Year (10)
		ProgCat (3)		

Suppose someone who is an outsider to the university wants to explore this data to find the key enrollment patterns. He has no prior knowledge of the data. Using existing OLAP exploration operations (like "drill-down" and "roll-up") he could launch the process of understanding the data by navigating through subsets of the cube viewed at various levels of aggregation. However, the process could be long and tedious involving perhaps repeated visits to the same parts of the data. Also, he could never really know if he has viewed everything that is interesting in the data.

We contrast this with the kind of focussed exploration possible with our tool. Not knowing anything else about the user, the tool starts out modeling the expected value of each non-empty cell to be the same. Starting from the top-level view, shown as the first screen-shot of Figure 1 he can ask for the most informative values to drill-down further. From the current view there are five drill-down paths along each of the five dimensions. Of these the most informative view returned to him is the Gender dimension. This shows that the fraction of males is 0.9 which is significantly higher than the average value of 0.5. The next surprising path is along the Category dimension which shows that the fraction of students in each category is also highly skewed. The members are sorted based on their information content. The most informative member is the `Indian` category that shows that the total number of Indians is 8223 — that is 82% of the total of 10083. This significantly departs from the expectation of having equal number of students in each of the nine categories. This cell is informative not only because it tells us that the fraction of Indians is high but also because it brings us closer to the actual values of the other members. When coupled with the knowledge in the first view that the sum of all values is 10083, we expect all other members to have a value of (10083-8223)/8 = 223 which is closer to their actual values. The next most informative dimension is the ProgCat dimension followed by the Year and finally the Dept dimension.

Another feature of the frontend, is an information-bar that displays the total information in the parts of the cube visited so far. For instance, from the five one-dimensional views of the cube in Figure 1 the user has acquired approximately 50% of the information content of the cube as shown by the half-filled status bar. For the remaining 50% the user needs to dig deeper. Depending on the user's interest, he could either continue with this guided drilling of data or ask for a summary of the remaining 50% of the information. We illustrate this second feature in the next scenario.

2.1.1 Scenario 2: Significant information in new data Consider another real-life data obtained from International Data Corporation (IDC) about the total yearly revenue for different software products from 1990 to 1994. The schema as shown below consists of four dimensions Product, Platform, Geography and Year and a three level hierarchy on the Product and Platform dimension. The values within bracket are the number of distinct members at that level of hierachy.

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)		

We now consider the case of a more informed user familiar with most of the detailed data except the last year 1994. The tool has kept track of this fact about the user. Based on the sales along time for each Product,Platform,Geography combination the user has formed an expectation of values in 1994. For instance, if for some combination there has been a steady increase along time, the user expects to see an increase, by the same ratio.

Category	(All)	Program	(All)
Gender	(All)	Dept	(All)
ProgCat	(All)	Year	(All)

	10083

Top-level view

	Gender	
	F	M
	987	9096

First most informative view along Gender dimension

Category								
Indian ...	Full time Sp...	Research A...	Institute fa...	Self finance...	QIP	External registr...	Foreign nati...	Project
8223	532	30	34	364	281	233	173	171

Second most informative view along Category dimension

		ProgCat	
	R	P	U
	1249	3890	4944

Third most informative view along ProgCat dimension

Year									
1989	1990	1991	1992	1993	1994	1995	1996	1997	1998
1877	982	843	837	858	886	997	899	993	911

Fourth most informative view along Year dimension

Electrical...	Mecha...	Civil E...	Chem...	Comp...	Meta...	Chem...	Earth...	Phy...	Mathe...	Hum...	Ind...	Bio-...	...	Corro..
1354	1318	1052	1046	839	749	513	297	276	238	109	172	137	...	89

Last view along Department dimension

Total percentage information in the top five views

Fig. 1 The drill down dimensions in order of information starting from the topmost view where all values are aggregated.

In general, the user can pick any time-series forecasting function to specify his expected values in 1994. In this example, let us assume that the user expects values in 1994 to be the average of the values in all the previous years. Let us also assume that the user has viewed the total revenue for 1994. This further changes the user's expectation of the values in 1994. In particular, he expects them all to sum up to the observed total for 1994. With this context about 1994 values suppose the user wants to quickly explore the remaining surprises or information in the 1994 data. The tool can return a summary of the key surprises in 1994 constrained to within N rows as specified by the user. The result is shown in Figure 2. In the figure, the last column marked "Expected" shows the values extrapolated from the user's context and the column before it shows the actual values. These are all cases that correspond to a significant drop or increase in 1994. For instance the last row shows that, based on prior knowledge the user expected the sales in 1994 for (DBMS Engines (Non-Object), Rest of World, Multiuser Mainframe IBM) to be 118.8 whereas the actual was just 10.99. The reported answer set also contains cells at the aggregate level which are able to present a representative summary of their children values. For instance, the middle row in the figure where all but the "Prod_Grp" level has a value of "(All)" summarizes all cells under the Prod_Grp "SySW". The last column "Ratio" indicates that for each member of this cell the expected value is 135% of the actual value. This provides the user with a compact summary

Prod_Gr...	...	Product	Geogra...	...	...	Platform	Year	Actual	Expected	% ratio
(All)	...	(All)	(All)	...	...	(All)	(All)	30,755.936	33,895.2...	110.2...
Soln	...	EDA	Weste...	...	...	Multiuser UNIX	1994	19.968	171.035	856.5...
Soln	...	Other ...	Unite...	...	...	Multiuser Minico...	1994	2.589	73.371	2,833...
SySW	...	(All)	(All)	...	...	(All)	(All)	10,256.613	13,857.0...	135.1...
Tool	...	CASE ...	Unite...	...	...	16-bit Windows/...	1994	1.769	69.156	3,909...
Tool	...	DBMS ...	Unite...	...	...	Multiuser Mainfra...	1994	90.307	378.362	418.9...
Tool	...	DBMS ...	Rest o...	...	...	Multiuser Mainfra...	1994	10.995	118.81	1,080...

Fig. 2 Informative regions returned to user who is familiar with the entire cube except for the last year 1994.

of the violators which cannot be extrapolated based on his experience of past data.

2.2 Scenario 3: Queries based on Ad hoc contexts

With this scenario we will show examples of how small contexts defined on the fly can be used to pose interesting queries. Consider again the Student data set above. Suppose a user wants to find the most atypical departmen-t based on their male female ratio. He can do so by first defining an ad hoc context consisting of two single dimensional totals along Gender and along Department dimensions respectively. Then, he can ask for the most informative department member in the Department*Gender plane. The de-partment so returned will be the most atypical department. Similarly, a query for the least informative department will return the department that is most typical. In Figure 3 we show the result for the student database. For the Software dataset of the previous section, he could ask for the most atypical Platform within a category, say "Unix M." based on the sales val-ues along time. In this case, the context will consist of total sales along time

	Gender	
Computer Science & Engineering ...	61	778
Earth Sciences	37	260
Electrical Engineering	86	1268
Humanities & Social Sciences	53	56

Fig. 3 The most typical department is "Computer Science" and the most atypical department is "Humanities and social sciences" in terms of the Gender distribution.

for "Unix M." and the total sales aggregated over time for platforms within the "Unix M." category.

In all the above scenarios we expect the user to continue to use the normal OLAP exploration tools. The new queries are designed to work along with the existing tools in an unobtrusive manner and are invoked only when the user needs the additional guidance that these queries offer.

3 Modeling expected values

In this section we discuss our basis of calculating the expected values of unseen parts based on some subset of the cube that the user has already visited and how we use these values to define the information content of an unseen cell.

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 cube in terms of the sum of values in subsets 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 $n = 2$ and assume that there are 16 total cells with four members each along the two dimensions: A (for names of geographies) and B (for names of products). The database thus consists of 16 real values as shown by the last column of Figure 4. The user visits various subsets and aggregates of the data during the course of his normal exploration of the data cube. We show an example of four views visited in sequence in the middle column of Figure 4. Suppose the user views only the sum of the 16 values in the cell (View 1). Let that sum be "160". Knowing nothing else about the data or the user's mind-set, what values can we assume for each of these cells? There are an infinite number of possible 16 values that sum up to 160. One possibility is to let the values of the first cell $p_0 = 160$ and the rest of the values p_1 to p_{15} be zero. Another is to let $p_0 = p_1 = 80$ and p_2 to p_{15} be zero. Both these alternative make rather bold statements based on the limited knowledge of the data. A more likely stance, the one that makes the least unwarranted assumptions, is to let each value be the same, i.e., $p_i = 160/16 = 10$. The first column in Figure 4 shows these expected values of the user.

Suppose the next visit is to the view that shows the totals along dimension A (View 2 in Figure 4). These immediately help the user refine his expectations of the 16 values as shown in the second table of the first column. Following the same logic, he can expect a value of $20/4 = 5$ each for products in Asia, $80/4 = 20$ each for products in Africa and so on. Suppose

Expected values

	OS	DB	Word	Prog
Asia	10	10	10	10
Afric	10	10	10	10
USA	10	10	10	10
UK	10	10	10	10

	OS	DB	Word	Prog
Asia	5	5	5	5
Afric	20	20	20	20
USA	12	12	12	12
UK	3	3	3	3

	OS	DB	Word	Prog
Asia	2.5	5	5	7.5
Afric	10	20	20	30
USA	6	12	12	18
UK	1.5	3	3	4.5

	OS	DB	Word	Prog
Asia	3	6	6	4.8
Afric	12	24	24.3	19.2
USA	3	6	6	33
UK	2	4	3.6	3

Visited views

View 1

	All
All	160

View 2

	All
Asia	20
Afric	80
USA	48
UK	12

View 3

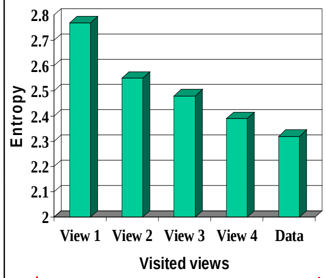
	OS	DB	Word	Prog
ALL	20	40	40	60

View 4

	Prog
Usa	33

Database

	OS	DB	Word	Prog
Asia	4	8	7	1
Afric	10	20	30	20
USA	5	9	1	33
UK	1	3	2	6



	OS	DB	Word	Prog
Asia	0	0	0	0.02
Afric	0	0	0.02	0
USA	0	0	0.03	0
UK	0	0	0	0.01

Information content

Fig. 4 Applying maximum entropy to calculate most likely values for an array with 16 members. The middle column denotes the sequence of values viewed by the user and the first (leftmost) column denotes the most likely expected values that confirms to these constraints.

the user visits a third view of the totals along dimension B as shown in Figure 4.

In this case, it is not all that obvious how we distribute the partial sums to derive individual values. Trying to satisfy one set of sums upsets the other set. The situation gets even more complicated when we add a fourth view that requires that the revenue of a member "Prog" of dimension B and "USA" of diemension A is 33 (View 4 Figure 4. This overlaps with both the totals along "USA" as required by the second view and the totals along "Prog". Fortunately, this is a classical problem that has found widely accepted answers in the Maximum Entropy principle [BPP96, GS85]. The final values as calculated using this principle are shown in Figure 4.

3.1 The maximum entropy principle

The maximum entropy principle dictates that we choose values of unknown cells so as to get the highest entropy while agreeing with everything that is known about the data. Entropy is a mathematical quantification of the information in data — data with highest entropy has the least amount of information. Thus, the maximum entropy principle attempts to assume as little as possible about the data. The entropy of a set of m values is defined as

$$H(p) = -\sum_{i=1}^{m} p_i \log p_i$$

Here p_i denotes the estimated value or probability of the ith 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 $\tilde{p}(C_i)$. The constraints can be overlapping in arbitrary ways. The final optimization problem is:

$$\max_p H(p) = \max_p(-\textstyle\sum_{i=1}^{m} p_i \log p_i) \quad \text{such that}$$

$$\forall C_i, \sum_j p_j I_{ij} = \tilde{\mathrm{p}}(C_i),$$
$$\text{where} \quad I_{ij} = \begin{cases} 1, \text{ if cell } j \text{ is included in } C_i, \\ 0, \text{ otherwise.} \end{cases}$$

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"*.

In the example above, we had several such constraints. In order to give a probabilistic interpretation to these values, we scale all values by the total of 160. Thus, the first constraint C_1 required that the sum of all cells be 1, i.e., $\tilde{\mathrm{p}}(C_1) = 1$. The second view gave four constraints correponding to the totals along each Geography. The second constraint is that the sum of values for Asia is 20/160, i.e., $\tilde{\mathrm{p}}(C_2) = 1/8$ and so on.

3.2 Finding the best values of p

The objective function $H(p)$ always has a unique solution as long as the constraints are consistent [PPL97]. Consistent constraints are those for which there exists at least one solution with all non-negative numbers that satisfies all constraints.

In most cases the objective function $H(p)$ cannot be solved in closed forms. 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.

$$p_j^\mu = \prod_{i=0}^{i=c} \mu_i^{I_{ij}} \tag{1}$$

We use p^μ to denote the class of p values that can be expressed in the above product form and p_j^μ is the expected value of the jth cell. For each constraint C_i there is a term μ_i. The first term μ_0 is a special normalization constant to ensure that the probabilities sum up to 1. We ensure this by adding a special constraint C_0 for which $I_{0j} = 1$ for all j and $\tilde{\mathrm{p}}(C_0) = 1$. The rest of the c constraints $C_1 \ldots C_c$ are explicitly defined by the user.

This formulation gives us a very general way of defining expected values without relying on any model or distribution. In fact, the multidimensional nature of the data is not relevant at all.

3.2.1 The Iterative scaling algorithm for finding best p There are several known variations of the iterative proportional fitting method for estimating the μ parameters. We present the most well-known form [Jay90] here.

Start with $\mu_i = 1$ for all constraints.

Update μ_0 so probabilities sum to 1

While convergence not reached

    For each constraint C_i $(0 \leq i \leq c)$

        Update μ_i by scaling with $\tilde{\mathrm{p}}(C_i)/p(C_i)$

        Recalculate expected values p using Equation 1

In the above algorithm, we start by first assigning all μ_i except μ_0 to 1. μ_0 is assigned so that all probabilities calculated using Equation 1 sum to

1. The result is that the starting values of all p_js is the same and equal to 1/(number of p values). This is the state of highest entropy. In the while loop we satisfy the additional constraints. In each iteration, all the constraints are cycled through in a round-robin manner. For each constraint i, we scale the μ_i values so that the ith constraint is satisfied. This might cause other constraints to be violated. The value $p(C_i)$ is obtained by summing up the p values of equation 1 calculated using the values of μ obtained from the previous iteration of the for-loop. The result of this update is that the p values included in the ith constraint get scaled by $\tilde{\mathrm{p}}(C_i)/p(C_i)$.

Convergence of the algorithm is checked by measuring the difference between the new and old values of p and stopping when the fractional d-ifference is not more than a user specified ϵ. The algorithm is guaranteed to converge to the optimal solution as long as all constraints are consisten-t [PPL97]. Constraints are said to be consistent if they do not violate each other's requirement, that is, where there exists a data cube of non-negative values that satisfies all the constraints. In our case, when the constraints correspond to the different aggregates viewed by the user, the consistency requirement is met.

In Figure 4 the values in the third and fourth rows of the first column show the expected values calculated after applying the constraints in the views above that row. For the third row one iteration was sufficient whereas for the fourth row the iterative procedure needed five iterations. The bar-chart in the last column of Figure 4 shows the entropy of the expected values after each of the four views get added. As more constraints get added, the entropy of the expected values get closer to the entropy of the data.

3.3 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 the new expected values p^{μ} by the maximum amount. Addition of a new constraint can potentially change the expected values of all cells of a cube. We choose that constraint that brings the resulting expected values closest to the actual values. We need a measure of the distance between the set of actual and the set of expected values of the cube. Distance between two distributions can be measured in a number of different ways. We chose the traditional Kullback-Leibler divergence criteria defined as:

$$D(\tilde{\mathrm{p}}||p^{\mu}) = \sum_{j} \tilde{\mathrm{p}}_j \log \frac{\tilde{\mathrm{p}}_j}{{p^{\mu}}_j} \tag{2}$$

In the above equation $\tilde{\mathrm{p}}_j$ denotes the actual value of cell j and p^{μ}_j denotes its expected value. When the actual and observed values are equal the distance is zero. In all other cases the distance is greater than zero as long as the sum of the observed and expected values is the same.

We now derive the information content of a new cell f using the above formula. Let p^C or p^c denote the expected values after the addition of the first C constraints of total size c and let p^{C+f} or p^{c+1} denote the expected values after adding a new $(c+1)$th constraint f. The information content of a constraint f is defined as the reduction in the distance between the actual and expected values after the addition of a new cell.

$$\text{Information content of } f = D(\tilde{\mathrm{p}}||p^C) - D(\tilde{\mathrm{p}}||p^{C+f}) \tag{3}$$

$$= \textstyle\sum_j \tilde{\mathrm{p}}_j(\log p_j^{c+1} - \log p_j^c). \tag{4}$$

In Figure 4, the table in the last row of the last column shows the information content of each of the sixteen cells after applying all the constraints in the middle column. Several cells show zero information content within two decimal places. These are the ones for which the actual and expected values are close. A comparison of the final expected values (in the last row of the first column) and the actual values (in the first row of the last column), shows that the least informative cells have very close actual and expected values. The largest information is in the cell 'USA'-'Word'.

We can simplify the above equation using equation 1. 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 μ_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 μ_{c+1}. Based on this assumption we write Equation 3 using results from Equation 1 and the iterative algorithm in 3.2.1. Thus, the simplified formula for the information content of a cell is

$$\text{Information content of } f = \textstyle\sum_j \tilde{\mathrm{p}}_j \log \mu_{c+1}^{I_{(c+1)j}}$$

$$\text{Information content of } f = \sum_j \tilde{\mathrm{p}}_j I_{(c+1)j} \log \frac{\tilde{\mathrm{p}}(f)}{p^c(f)} \tag{5}$$

3.4 Justification for the use of the proposed model

The goal of our tool is to highlight to a user regions of a cube that he will find most informative or surprising. There are several ways of achieving this objective.

A simple approach, that which is practised by several OLAP vendors, is to let the user define the criteria for what is surprising. The system applies the critiria on each cell and highlights the ones that meet the criteria. Examples of such criteria are: sales dropping below 5% of previous year's or revenue varying by more than 10% from forecasted revenue. These approaches are effective in limited domains but the burden is on the user to make the criteria explicit. This could result in too many cells being highlighted as surprising. Also it provides no discovery element to the user. The user has to foresee in advance all ways in which he could find something surprising and explicitly express these to the system.

A second more automated approach is to choose beforehand a class of models that can predict the measures as a function of the other dimension members. The model is used to compute the expected measure value for each cell. The expected values are compared with the actual measure values and large differences between the two values results in the cell being called informative or surprising. This method can be made arbitratrily complicated by choosing the model accordingly. An examples of a simple models is an autoregressive time series model where the measure values along time are used to predict a future value. This approach only takes into account the effect of a single dimension, time. Richer loglinear models that attempt to incorporate the dependencies between dimensions and the effect of other dimensions (e.g., geography) are also possible as discussed in [SAM98]. The disadvantage of all model-based approaches is that the model once chosen is static and does not take into account the dynamic context of the user as he learns more and more about data.

The maximum entropy approach provides an elegant way of incorporating a user's partial knowledge of the data. At some level, human knowledge is "subjective" and even in the first approach the tools provided by the system might not enable him to precisely define the surprise criteria so as to capture his entire subjective context. We capture human knowledge in terms of what views of the data the user has already visited. In terms of modeling power, Maximum Entropy is powerful because it allows us to model dependencies of multiple overlapping events without explicitly modeling the structure of their interaction. Second, it makes no assumptions about the data. It just attempts to infer the best it can without making any undue assumptions. Entropy is a mathematical measure of ignorance, higher entropy implies higher ignorance. The maximum entropy approach allows us to assert the maximum ignorance while fully accounting for everything that is known.

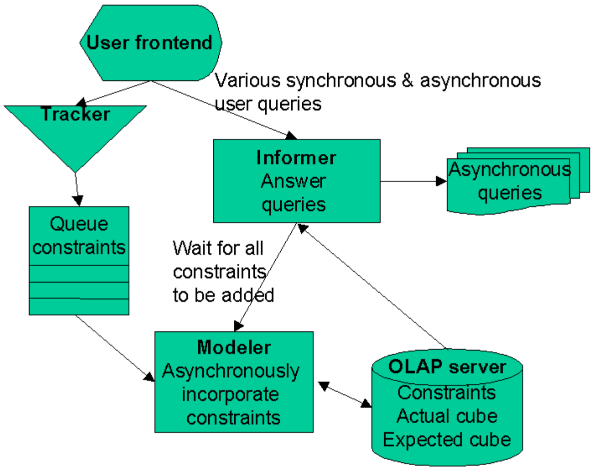


Fig. 5 System architecture

4 Implementation and computation issues

In this section we discuss how we implement the techniques of the previous section and how we address the computation issues that arise in applying the maximum entropy principle for informative explorations of OLAP data.

Figure 5 gives a block diagram of the system. The frontend is any multidimensional OLAP exploration tool. We view the measures as the probability values of the previous section. For measures that denote counts like the number of units sold or the total inventory, a straight normalization by the total provides the right mapping. Other measures such as sales can be converted to this form by dividing with the unit price. The aggregate function is assumed to be additive like sum or count. This is the most common aggregate function. Non-additive functions like “max” or “min” cannot be handled in this framework. The users navigation takes him through sums of measures of different subsets of the cube at different aggregation levels.

Each of these visited views forms constraints to the system. Alternately, the user can choose to add such constraints explicitly instead of relying on automatic monitoring by the Tracker. Each *view* of the cube corresponds to a collection of constraints — one corresponding to each displayed cell in the view. For instance, consider a cube with four dimensions A, B, C and D. 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. We use the lower case letter a_i to denote the ith 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.

The constraints above have the form: sum of a set of detailed measures of the cube is a constant. We call these "observed" constraints. A second kind of constraint is obtained by explicit input from the user about what values he expects to see at some subset of the cube. We call these "tentative constraints". An example of tentative constraints was introduced in Scenario 2 of Section 2.1.1 where the user explicitly stated expected values for the 1994 year. An important distinction between tentative and observed constraints is that the set of observed constraints are always consistent whereas user-supplied tentative constraints can be inconsistent with another observed or tentative constraint. The observed constraints have been obtained by adding up values from an existing cube, so there is an existential proof that all these constraints can be simultaneously satisfied. In contrast, tentative constraints are obtained by the user explicitly specifying aggregated values which might be inconsistent if the user is not careful.

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 constraints and second is expected values based on the context so far. This way the user can regain the same context next time he logs on. When a new constraint is submitted we update the values of expected probabilities using the iterative algorithm given in 3.2.1. We do not explicitly ever store the μ values. The expected values are scaled directly. Tentative constraints are treated a bit differently than observed constraints. Tentative constraints do not participate in the iterative scaling algorithm. The first time a tentative constraint is added we set the initial expected values just once to satisfy these constraints. Later the observed constraints are iterated over to reach a convergent solution. Since only the consistent observed constraints (and not the tentative constraints) participate in the iterative refinement we expect the solution to be convergent.

All processing and storage is done in units of "views" rather than constraints. Thus, in the constraint table we store a row for each view. Similarly, the iterative algorithm simultaneously updates all expected values that are part of the same view. By making the "view" instead of its individual constraints as the unit of processing, performance is enhanced albeit at the cost of higher code complexity.

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 of these are computationally expensive procedures. We next discuss a collection of optimizations that we applied on these methods to make them efficient on large OLAP datasets. In Section 4.1 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 4.2 we present optimizations for getting answers to find the most informative regions.

4.1 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.

4.1.1 Asynchronous batched computation The first architectural level change we propose is batching updates due to multiple constraints. When the user submits a view, the request is queued and the user call returned. 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.

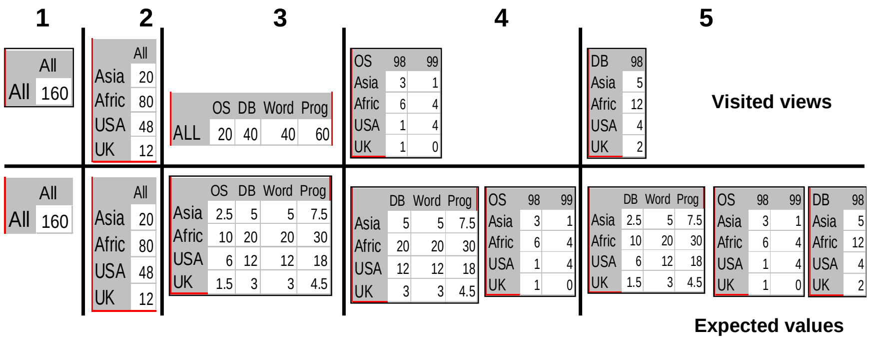


Fig. 6 The upper row of tables denotes the five visited views of the cube and the lower row shows the optimized expected cube as stored in the database. The cube consists of three dimensions: A, the first dimension as shown in view 2, B the second dimension as shown in view 3 and a third dimension C with two members as shown in view 4.

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, when there are new constraints in the queue we run just one step of the algorithm and apply just that new constraint. When no more constraints are waiting we invoke the iterative algorithm to refine the expected values. In Figure 5 the queue of submitted constraints shows the asynchronous processing we follow.

4.1.2 Optimized representation This optimization relates to how we store the 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.

Consider the example two dimensional cube of Figure 4 with an additional dimension C along time with two members. Thus the cube has $16 \times 2 = 32$ total cells. In Figure 6 we show a set of five visited views on the top row and the corresponding optimized expected cube that is stored after each view on the bottom row. After the first view, the expected cube stores just one cell, containing the sum 160 and the count 32. The expected value of all cells is $\frac{160}{32} = 5$. Thus, instead of storing the expected cube as 32 cells we store just one entry. The second view is along dimension A as shown in the second column of Figure 6. All detailed cells with the 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 ABC. The expected value of individual detailed cells is obtained by dividing the expected values at the A level by the number of its children at the detailed level ABC. The third view is along dimension B, thus all cells with the same value of dimensions A and B will have the same expected value, hence we store the expected values at the AB level as shown in the third column of Figure 6.

This optimized representation makes the addition of a new constraint more complicated. Every time a new constraint is added, we might need to de-aggregate the level to which some expected value regions are stored.

For instance, in our previous example when the third view is added we find that A needs to be expanded to AB. This expansion is done as follows. We extract from the data cube the actual values and counts for cells at the AB level and join with the expected cube at the A level to get the new scaled expected values at the AB level. The iterative algorithm is run on these values to get the final expected values shown in the Figure. When a fourth view Ab_jC (b_j = 'OS') is added, we need to expand only a subset of the entries. As shown in the Figure we retain all the other product names at the original AB level and only expand the entries for 'OS' to dimension C. When a fifth view Ab_lc_k (b_l ='DB' and $c_k = 1998$) is added, a slightly different approach is followed. We retain the column for 'DB' at the AB level, but subtract from it the expected values and counts of year '1998'. The expected values for product 'DB' and year '1998' are stored at the most detailed level separately. Notice that the expected values for DB are halved when we go from view 4 to view 5 because one of its two members is stored separately.

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

4.1.3 Optimizing the 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.

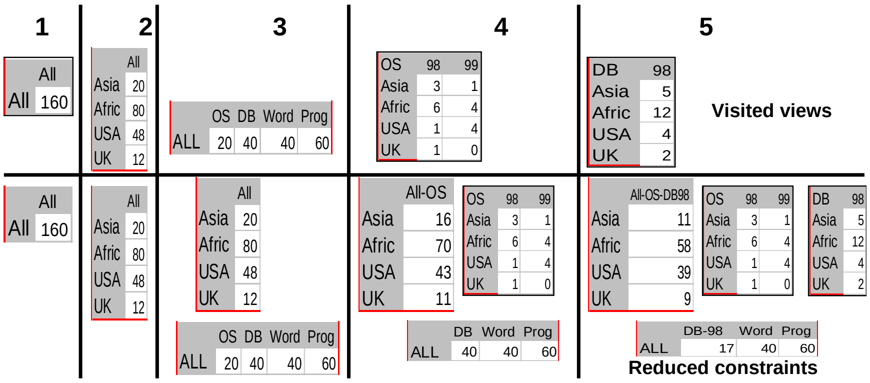


Fig. 7 The upper row denotes the five visited views of the cube and the lower row shows the constraints after removing subsumed constraints.

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, 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 The maximum entropy method can handle any amount of overlap amongst constraints. However, highly overlapping constraints require more iterations for convergence. We propose rewriting constraints whenever possible to reduce such overlap. When a constraint C_1 is subsumed by another constraint C_2 we propose modifying C_2 so as to exclude all the elements of C_1 from it.

The procedure is explained next through a running example in Figure 7. The set of constraints as they arrive are present in the top row and in the second row we have the reduced set of constraints after removing subsumed constraints as implied by this optimization. The first constraint is that the sum of all values is 160. The second view is totals along the first dimension A. This view adds four constraints $C_1 \ldots C_4$ one for each member a_i of A. They together subsume the first constraint C_0. So, we delete C_0. The next view is dimension B contributing four more constraints $C_5 \ldots C_8$. No constraint in this view is subsumed by any of the four existing cosntraints, so we add these constraints without any change. The fourth view is of the form Ab_jC where for a fixed value 'OS' of the second dimension B, we have values for different combinations of dimensions A and C. These constraints together totally cover the constraint C_5 of the third view, hence we remove C_5 as shown in the second row of the fourth column. Constraints $C_1 \ldots C_4$ of the first view that sum over all products get rewritten to subtract values for 'OS' as shown in the Figure. The fifth view is of the form Ab_jc_k where for a fixed member 'DB' along dimension B and '98' along dimension C we have values for different members of A. These constraints are subsumed by C_6 of the third view and $C_1 \ldots C_4$ of the second view. Hence we update them to subtract the subsumed part. Notice that for 'OS' we could remove the corresponding constraint C_5 whereas for 'DB' C_6 could not be removed because not all members of the third dimension are present in the fifth view.

This optimization is very useful for OLAP data because the exploration pattern is such that often a user starts at an aggregated level say AB and follows it up with a selection on a previously aggregated dimension say C. This adds a constraint of the form ABc_j that is subsumed by AB. Rewriting thus significantly reduces the number of iterations because of the reduction in the overlap between constraints. Also, the implementation of these dynamic subsumption relations is made easy because we never explicitly store the μ values. The p values are scaled directly when a new constraint needs to be applied.

Prune constraints The algorithm of Section 3.2.1 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 the 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 δ_j denote this maximum change on a cell due to a constraint C_j. For a constraint C_k at the kth 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 the 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 $20/1000 = 0.02$.

4.2 Optimizing queries on information content

We now address how the Informer uses the expected values to answer various user queries on the information content in unseen parts of the cube. We review the different kinds of queries introduced in Section 2, provide a formal definition of each and present algorithms for efficiently processing them. We first recap and refine the definition of the information content of a cell.

4.2.1 Information content of cell In Equation 5 we presented a simlified definition of the information content of a cell that ignored its effect on the coefficients of other constraints. We next discuss how we remove this assumption when measuring the information content of a collection of cells simultaneously. The information in two cells might be less than the sum of the information in each calculated using Equation 5. Consider the example of Figure 1. After the first view we know that the total enrollment is 10083. In the second view the two constraints: first that the total total for "M" is 9096 and second that the total for "F" is 987, are both individually quite informative. However, knowing one makes the other redundant when combined with the first view. We discuss how we refine Equation 5 to handle such interaction. We first consider a single constraint and then more than one constraint. When defining the information content of a cell f Equation 5 considers only the change in expected value due to the coefficient of that cell. It ignores the change in the coefficients of all the other constraints due to the addition of cell f. We refine this definition as follows: Let C_i and μ_i denote the corresponding constraint and coefficient respectively. For cell f, find the closest parent of the cell in the list of constraints. Let C_g and μ_g denote the constraint and coefficient of the parent cell respectively. After applying the remove-subsumed optimization of Section 4.1.3, constraint C_g gets modified so as not to include the elements of C_f. Thus, we scale μ_g as follows: $\frac{\tilde{\mathrm{p}}(C_g)-\tilde{\mathrm{p}}(C_f)}{p(C_g)-p(C_f)}$. The total change due to cell f now can be summarized as follows:

$$\mathrm{info}(f) = \sum_j \tilde{\mathrm{p}}_j I_{fj} \log \frac{\tilde{\mathrm{p}}(C_f)}{p(C_f)} + \sum_j \tilde{\mathrm{p}}_j I_{gj} \overline{I_{fj}} \log \frac{\tilde{\mathrm{p}}(C_g) - \tilde{\mathrm{p}}(C_f)}{p(C_g) - p(C_f)} \qquad (6)$$

In the second term, the indicator $\overline{I_{fj}}$ stands for the negation of I_{fj} and serves to exclude all elements that are included in the constraint C_f.

We next discuss how we measure information content of a set of k cells denoted as $F = \{f_1, \ldots, f_k\}$. We assume that each of these cells are disjoint i.e., they do not overlap in any detailed value. This is true, for instance, when the cells correspond to different members of the same dimension. Let g denote the most detailed common parent of these cells. The information content of F is then defined as follows:

$$\text{info}(F) = (\sum_{i=1}^{k}\sum_{j} \tilde{\text{p}}_j I_{f_i j} \log \frac{\tilde{\text{p}}(C_{f_i})}{p(C_{f_i})}) + \sum_j \tilde{\text{p}}_j I_{gj} \overline{I_F} \log \frac{\tilde{\text{p}}(C_g) - \sum_{i=1}^k \tilde{\text{p}}(C_{f_i})}{p(C_g) - \sum_{i=1}^k p(C_{f_i})} \quad (7)$$

In the example of Figure 1 just discussed, the above formula will yield the same value of the information content for each of the three cases: first where the set is {"M"}, second where the set is {"F"} and third where it is {"M", "F"}. In the third case, the second term of the above Equation becomes 0.

We next discuss how to answer the four different queries discussed in Section 2 using the above definition of information content.

4.2.2 Most surprising drill-down path Starting from an aggregated view of the cube a user can query for the most informative drill down path. For instance, in the first scenario in Section 2.1 a user starting at the highest aggregation level wanted to find which of the five drill-down paths are most informative. The input for this query is one view v_q of the cube at some aggregated level and the goal is to search its neighborhood for the most informative view. The neighborhood of v_q consists of all one and two dimensional views that differ from v_q in not more than two positions and is at a more detailed level than v_q.

A natural way to define the information content of a view is to measure the information conveyed by all cells of the view combined. The problem with this definition is that it is biased towards views with a large number of cells. One way to get around this bias is to use the ratio of the total information in the view and the total number of cells. This has the opposite effect of penalizing large views because in any large view there will always be many cells that are not informative. We chose a middle ground by selecting the view with the highest total information in its first k most informative cells. The number k is configurable by the user. For the first scenario of Section 2.1 k was set to 1. These queries are not computationally intensive because only a restricted subset of the data needs to be searched. We expect such queries to be more frequent at the top levels of the cube when the user is relatively unfamiliar with the rest of the cube.

4.2.3 Sort by information content Once a view is selected using the previous query, we need to sort the members based on their information content. We measure the information content of each cell assuming that the cell prior to it in the sorted order is already added as a constraint. This requires an $O(n^2)$ algorithm where n is the number of members along that dimension. The algorithm makes n iterations picking the ith element in the sorted order of information content in the ith iteration. At each iteration i we recompute the information content of the remaining $n - i$ constraints assuming the $i-1$ constraints chosen before it are already known as constraints. Since n is typically small, the quadratic complexity of the algorithm is tolerable.

4.2.4 Percentage information accrued This query returns the percentage information of all the constraints submitted by the user so far. We evalutes this query as follows. We first fix two extremes. The first extreme is where the user has full information, when the expected value p is equal to the observed values $\tilde{\mathrm{p}}$. The second extreme is zero information, when the expected value of each cell is the same, say $\frac{1}{n}$. The distance $D(\tilde{\mathrm{p}}||p)$ between the actual values $\tilde{\mathrm{p}}$ and expected values p, calculated using Equation 2, in the first case is zero. In the second case it is $-\frac{1}{n}\sum_j \tilde{\mathrm{p}}_j \log \tilde{\mathrm{p}}_j$. The current distance value $D(\tilde{\mathrm{p}}||p)$ expressed as a percentage of the distance range $\frac{1}{n}\sum_j \tilde{\mathrm{p}}_j \log \tilde{\mathrm{p}}_j$ is the information measure returned to the user.

4.2.5 Most informative N row summary In this case, the goal is to find the N most informative constraints from anywhere in the cube including cells from aggregate levels as shown in the example of Figure 2. We expect these queries to be 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, these queries are more challenging since they require searching the entire cube. Another source of complexity is that since the constraints are allowed to be from anywhere in the cube they interact with each other in arbitrary ways and we need to measure the information in all of them simultaneously. For example, including a constraint of the form $a_i b_j$ changes the information content of constraint $a_i b_j c_k$ and vice versa.

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 up to 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 constraints. Even with this restriction finding the optimal solution is non-trivial because of the interactions between subsumed constraints.

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].

5 Experiments

We evaluate our system along two important metrics: performance and quality of data exploration.

System platform This tool is 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 $\mathbf{i^3}$ 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 JDBC to access data. Both of these engines provide advanced indexing and materialized aggregate views for efficient processing of OLAP queries. The data is stored in a star-schema. For Oracle 8i, we use the appropriate "Dimension" and "Hierarchy"[Ora99] constructs to better facilitate query rewrites. For the Microsoft OLAP server we use MDX via OLE DB for OLAP [Mic98b]. 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. The experiments were done on a PC with a 333 MHz Intel processor, 128 MB of memory and running Windows NT 4.0. DB2 UDB 6.1 database was used to process the queries.

Dataset 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 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 level 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 brackets 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 dataset introduced in Section 2 has just 4560 cells at the detailed level. This is small by OLAP standards. We therefore do not use this dataset for timing studies. We use it only for qualitative assessments of our method because the dataset is real.

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

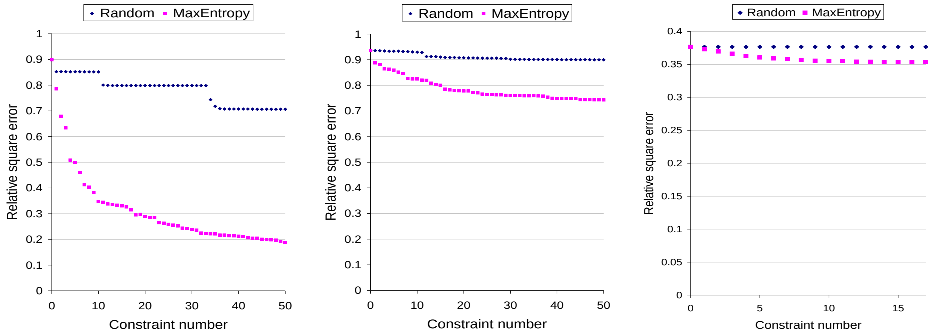
Store	Customer	Product	Promotion	Time
Name (24)	City (109)	Name (1560)	Media type (14)	Month (24)
State (10)	State (13)	Subcategory (102)		Quarter (8)
Country (3)	Country (2)	Category (45)		Year (2)
		Department (22)		
		Family (3)		

5.1 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. 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 by following two random steps: First, the user randomly selects one dimension d_i from the two dimensions that are currently either row or column and assigns a fixed value to it. The fixed value is either one of its members or "All". Second, he selects another dimension d_j from the set not currently forming a viewing dimension to replace d_i. This yields a new view of the cube from which the user can move to a neighboring view using the same procedure. We present an example of such a sequence of views for the three dimensional data cube of Figure 6. Initially, the user is at the topmost level denoted by (All, All, All), i.e., all dimensions are aggregated. We randomly pick dimension B to drill further. Thus, the next visited view will be (All, B, All), followed by another random drill-down to dimension A. The user is now viewing the data in the context of two dimensions (A, B, All). For the fourth view, we randomly select member 'OS' of dimension B and use dimension C to replace B as the viewing dimension. This yields the view (A, 'OS', C). For the fifth view, we randomly decide to aggregate data along A and make B one of the viewing dimensions. This is the view (All, B, C). From here the user could look for details along A and B for a fixed value '98' of dimension C resulting in a view $(A, B, 98)$.

5.2 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, we simulate a random exploration similar to the workload described earlier. For each view that the user visits, the constraints in that view are submitted to the system. In the second case, the user starts by submitting the top-level aggregate as a constraint. Next, he asks the system for the most informative cell, submits that as the second constraint, and asks the system again for the next most informative cell and so on. Each time the user submits the constraint that the system returns as the most informative. This is meant to simulate an "optimal" exploration in terms of the information content of each visit. In Figure 8 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 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%



(a) Student data. (b) Software revenue data. (c) Grocery sales data.

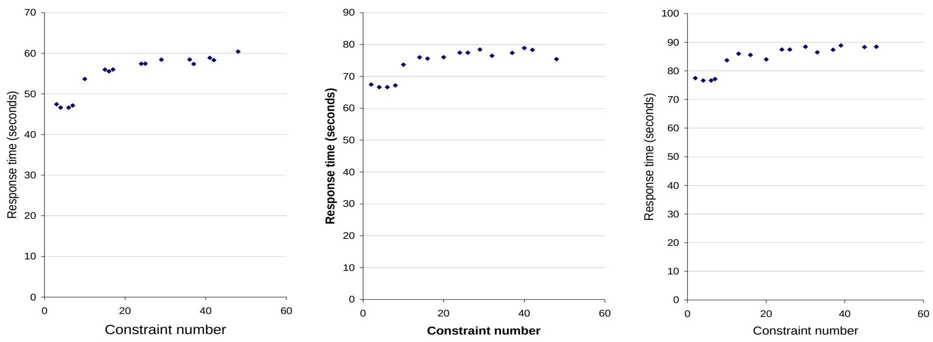
Fig. 8 Change in error as constraints get added. The curve 'Random' shows error for a random set of constraints and the curve 'MaxEntropy' shows error with the most informative constraints

of the information content. The error for the Grocery sales data reduces at a much slower pace. The first constraint corresponding to the sum of all values captures more than 60% of the total information. Subsequent constraints add just 2% to the total information content. The reason, we later found, was that the dataset was synthetically generated using perhaps a uni-modal distribution.

These experiments show the power of the new focussed exploration in getting to the key informative regions of the cube efficiently.

5.3 Timing measurements

In this section we present timing results. 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 the asynchronous processing as discussed in Sec-



(a) Software revenue data. (b) Grocery sales data. (c) OLAP benchmark.

Fig. 9 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.

tion 4.1.1. The main concern is response time of the second part. However, the response time of the second part is affected by the efficiency of the first part because we need to ensure that all prior constraints are processed and expected values upto-date before finding the informative cells. In Section 5.4 we study the efficiency of the first part.

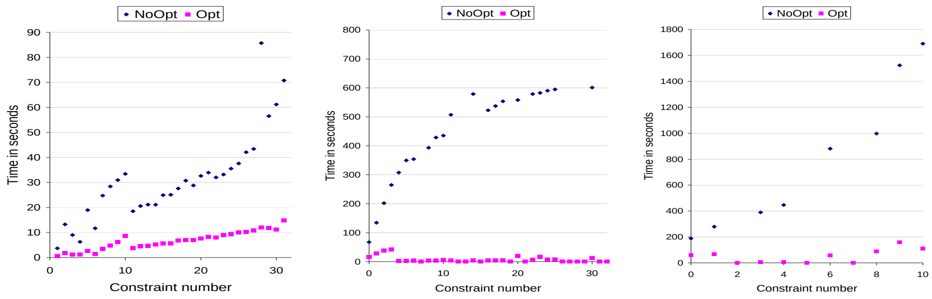
First we show overall response time on the top-N informative queries. For response time measurements we augment the workload in Section 5.1 with timing information. The time spent on a view of the cube is set to be a function of the number of cells in the current view. We assume that per cell the user 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 navigations of the cube.

Figure 9 shows the response time for the top-N informative constraints as a function of the number of constraints after which the query is posed. Even for our largest dataset which is the OLAP benchmark with 1.3 million tuples we were able to return the best answer within 2 minutes. This 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. Typically, with servers having memory in the gigabytes range, we expect processing speed, rather than I/O to be the bottleneck because the iterative algorithm would operate on cached in-memory data in such cases. Also, for large cubes where interactive response gets harder, we expect the user to submit the query in asynchronous mode.

5.4 Experiments on improving the iterative algorithm

In this section we report on further experiments to evaluate the impact of various optimizations discussed in Section 4.1 for improving the iterative algorithm.

Improvement due to optimized representation We demonstrate the impact of the optimized representation discussed in Section 4.1.2 by measuring total time taken by the iterative algorithm as constraints get added. The datasets and workload settings are as discussed earlier. In the graphs in Figure 10,

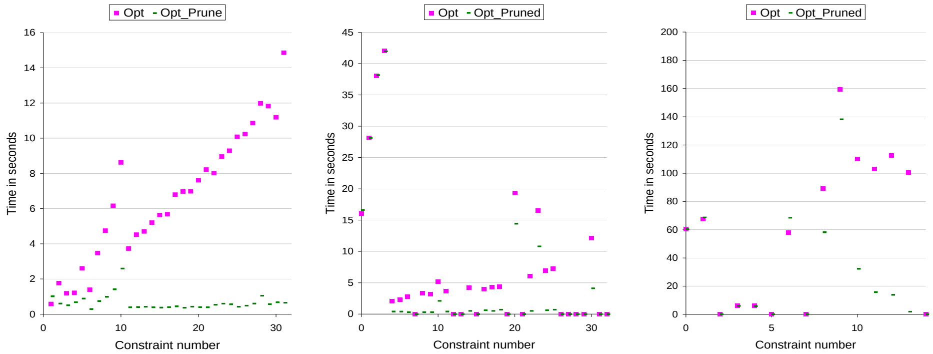


(a) Software revenue data. (b) Grocery sales data. (c) Olap benchmark.

Fig. 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.

the X axis is the number of constraints in the order of arrival and the Y axis is the total time for the iterative process to propagate the effect of the constraint. We plot two graphs, one for the optimized representation (marked "opt") and second for the unoptimized detailed representation (marked "noopt"). We observe a factor of five improvement in total time for the software data and even greater, between factors of 10 and 100, for the other two 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 could be completed in half a minute with the optimized representation.

Improvements due to reducing overlap and pruning constraints We demonstrate the impact of the optimizations of Section 4.1.3 on the iterative algorithm. The setup and the axes are the same as in Section 4.1.2. Data is assumed to be stored in the optimized representation of Section 4.1.2.



(a) Software revenue data. (b) Grocery sales data. (c) Olap benchmark.

Fig. 11 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.

In Figure 11 for each dataset we plot two graphs — one for the iterative algorithm after including the remove-subsumed and prune-constraints optimizations (marked "opt-pruned) and one without these two optimizations (marked "opt"). The graphs show another factor of two reduction in total time with these optimizations. The improvement is lower in early stages when the number of constraints is small and increases as more constraints get added.

6 Related work

The work reported here is part of our continuing $\mathbf{i^3}$ project [SS00] 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 $\mathbf{i^3}$ project is to first investigate how and why analysts currently explore the data cube and next to 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 definitions 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 can 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 4.2 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. It was first used in the 19th century by J. Willard Gibbs to infer the thermodynamic properties of physical systems from their mean energy values without making any unwarranted assumptions. In the 20th century, E.T. Jaynes applied maximum entropy to the more general problem of acquiring knowledge from data. 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, PMS00] presents a more focussed application of Maximum Entropy to the problem of frequent itemset mining. [PMS00] use a slight variation of the iterative algorithm we presented in Section 3.2.1. The algorithm updates μ_0 along with μ_i for each iteration of the for-loop so that all values not belonging to μ_i get scaled to take up the deviation from one of the sum of all ps. We did not find this method efficient. In particular, when all constraints are disjoint then after applying the remove-subsumed optimization of Section 4.1.3 our iterative procedure will converge after one iteration whereas the algorithm in [PMS00] needs more iterations even for simple examples.

7 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.

Acknowledgements I would like to thank the anonymous reviewers for their detailed comments that greatly helped to improve the clarity of the presentation.

References

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

[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.html`.

[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.

[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/`.

[GS85] S. Guiasu and A. Shenitzer. The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1):42–48, 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*, pages 420–431, 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*, pages 1–13. Kluwer, 1990.

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

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

[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.

[Ora99] Oracle. *Oracle 8i Database Administrator's Guide*, 1999.

[PMS00] D. Pavlov, H. Mannila, and P. Smyth. Probabilistic models for query approximation with large sparse binary data sets. In *Sixteenth Conference* *on Uncertainty in Artificial Intelligence*, 2000.

[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)*, pages 42–53, 1999.

[Sar00] S. Sarawagi. User-adaptive exploration of multidimensional data. In *Proc. of the 26th Int'l Conference on Very Large Databases (VLDB)*, pages 307–316, 2000.

[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`.