Sampling Cube: A Framework for Statistical OLAP Over Sampling Data[*]

Xiaolei Li, Jiawei Han, Zhijun Yin, Jae-Gil Lee, Yizhou Sun
Department of Computer Science, University of Illinois at Urbana-Champaign
Urbana, IL, USA
xli10@uiuc.edu, hanj@uiuc.edu, zyin3@uiuc.edu, jaegil@uiuc.edu, sun22@uiuc.edu

ABSTRACT

Sampling is a popular method of data collection when it is impossible or too costly to reach the entire population. For example, television show ratings in the United States are gathered from a sample of roughly 5,000 households. To use the results effectively, the samples are further partitioned in a multidimensional space based on multiple attribute values. This naturally leads to the desirability of OLAP (Online Analytical Processing) over sampling data. However, unlike traditional data, sampling data is inherently uncertain, *i.e.*, not representing the full data in the population. Thus, it is desirable to return not only query results but also the confidence intervals indicating the reliability of the results. Moreover, a certain segment in a multidimensional space may contain none or too few samples. This requires some additional analysis to return trustable results.

In this paper we propose a *Sampling Cube* framework, which efficiently calculates confidence intervals for any multidimensional query and uses the OLAP structure to group similar segments to increase sampling size when needed. Further, to handle high dimensional data, a *Sampling Cube Shell* method is proposed to effectively reduce the storage requirement while still preserving query result quality.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

General Terms

Algorithms, Design, Experimentation

1. INTRODUCTION

Data warehousing and online analytical processing (OLAP) are the primary components of decision support systems. Typically, the source data of OLAP are complete and consolidated historical data collected from many parts of the organization. This means that the OLAP query processor has access to *all* the relevant data and can produce an accurate answer anywhere in the multidimensional space. Data cubes [10] are often used to pre-compute multidimensional aggregates and thus provide faster runtime performance.

In many applications, however, the complete source data is not available. Only a **sample** of the population is available for study. Consider the following example.

Example 1 (SAMPLING DATA). Nielsen Television Ratings *in the United States are the primary source of measuring a TV show's popularity. Each week, a show is given a rating, where each point represents 1% of the US population. There are over 100 million televisions in the United States, and it is impossible to detect which shows they are watching every day. Instead, the Nielsen ratings rely on a statistical sample of roughly 5000 households across the country. Their televisions are wired and monitored constantly, and the results are extrapolated for the entire population.*

This example shows a typical use of sampling data. In many real world studies about large populations where the data collection is required, it would be too expensive and often simply impossible to gather the relevant information from *everyone* in the population. Nonetheless, multidimensional analysis must be performed with whatever data is available.

Example 2 (OLAP ON SAMPLING DATA). *Advertisers are major users of TV ratings. Based on the rating, they can estimate the viewership of their advertisements and thus pay appropriate prices. In order to maximize returns, advertisers want the maximum viewership of their target audience. For example, if the advertised product is a toy, the advertiser would want a TV show with children as its main audience. As a result, advertisers demand ratings be calculated in a multidimensional way. Popular dimensions include age, gender, marital status, income, etc.*

To accommodate the multidimensional queries, attributes are recorded at the sampling level, *e.g.*, a recorded viewership for television show X might be attached to a "married male with two children." This leads directly to **OLAP on multidimensional sampling data**. Compared to traditional OLAP, there is a subtle and yet profound difference.

[*]This work was supported in part by the U.S. National Science Foundation NSF IIS-05-13678 and BDI-05-15813, and by the Boeing company. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
Copyright 2008 ACM 978-1-60558-102-6/08/06 ...$5.00.

In both cases, the intent or final conclusion of the analysis is on the population. But the input data are very different. Traditional OLAP has the complete population data while sampling OLAP only has a minuscule subset. Table 1 shows a summary of the differences.

Input Data	Analysis Target	Analysis Tool
Population	Population	Traditional OLAP
Sample	Population	*Not Available*

Table 1: Two models of OLAP application

The question is then "Are traditional OLAP tools sufficient for multidimensional analysis on sampling data?" The answer is "No" for several reasons. First is the *lack of data*. Sampling data is often "sparse" in the multidimensional sense. When the user drills down on the data, it is very easy to reach a point with very few or no samples even when the overall sample is large. Traditional OLAP simply uses whatever data is available to compute an answer. But to extrapolate such an answer for the population based on a small sample could be dangerous. A single outlier or a slight bias in the sampling can distort the answer significantly. For this reason, the proper analysis tool should be able to make the necessary adjustments in order to prevent a gross error. Second, in studies with sampling data, statistical methods are used to provide a measure of reliability on an answer. Confidence intervals are usually computed to indicate the quality of answer as it pertains to the population. Traditional OLAP is not equipped with such tools. And lastly is the challenge of high dimensionality in the input data. While common to many other problems, sampling data analysis offers some new challenges such as the relationship between the quality of samples and an extremely large number of subspaces.

Example 3 (Usage Example). *To give a more concrete application of multidimensional sampling data, consider a retail outlet trying to find out more about its customers' annual income levels. In Table 2, a sample of the survey data collected is shown*[1]*. In the survey, customers are segmented by four attributes, namely `Gender`, `Age`, `Education`, and `Occupation`.*

Gender	Age	Education	Occupation	Income
Female	23	College	Teacher	$85,000
Female	40	College	Programmer	$50,000
Female	31	College	Programmer	$52,000
Female	50	Graduate	Teacher	$90,000
Female	62	Graduate	CEO	$500,000
Male	25	Highschool	Programmer	$50,000
Male	28	Highschool	CEO	$250,000
Male	40	College	Teacher	$80,000
Male	50	College	Programmer	$45,000
Male	57	Graduate	Programmer	$80,000

Table 2: Sample survey data

First, the manager tries to figure out the mean income level of programmers in their customer base (i.e., `Occupation` = Programmer). The mean income for them is $55,400 with a 95% confidence interval of ± $12,265. This seems like a reasonable answer to the manager. Next, the manager queries for the mean income of teachers in the 30–40 age range. Although the data only contains one sample matching this criterion, an "expanded" answer of $85,000 ± $5,657 with 95% confidence is returned and the manager is satisfied.

1.1 Contributions

To this end, this work proposes a new framework, called **Sampling Cube**, which adds the following features to the traditional OLAP model: (1) Calculations of point estimates and confidence intervals for all queries are provided. Algebraic properties of the measures are exploited to make the calculations efficient. (2) When a query reaches a cell with too few samples, the query is "expanded" to gather more samples in order to improve the quality of the answer. The expansion takes advantage of the OLAP structure to look for semantically similar segments within the queried cuboid and also nearby cuboids. Two sample hypothesis tests are performed for expansion candidates, and the ones that pass are merged with query segment to produce the final answer. (3) Lastly, to handle the high dimensionality problem, a **Sampling Cube Shell** method is proposed. Instead of materializing the full sampling cube, only a small portion of it is constructed. But unlike other cube compression schemes, the selection is based on the quality of sampling estimates. In tests with real world sampling data, the framework is shown to be efficient and effective at processing various kinds of queries.

1.2 Paper Organization

The rest of the paper is organized as follows. In Section 2, formal definitions of the problem are given. Section 3 describes the whole sampling cube framework of efficient aggregation and query expansion. Section 4 describes the sampling cube shell with optimizations for high dimensional data. Section 5 shows experimental results with respect to query efficiency and effectiveness. Related work is discussed in Section 6, and the paper concludes in Section 8.

2. DEFINITIONS

2.1 Data Cube Definitions

Before giving the proper problem definitions, we will review the data cube model. Given a relation R, a *data cube* (denoted as C_R) is the set of aggregates from all possible group-by's on R. In an n-dimensional data cube, a cell $c = (a_1, a_2, \ldots, a_n : m)$ (where m is the cube measure on some value) is called a k-dimensional group-by cell (*i.e.*, a cell in a k-dimensional cuboid) if and only if there are k ($k \leq n$) values among $(a_1, a_2, \ldots, a_n)$ which are not $*$ (*i.e.*, all). Given two cells c_1 and c_2, let V_1 and V_2 represent the set of values among their respective $(a_1, a_2, \ldots, a_n)$ which are not $*$. c_1 is the *ancestor* of c_2 and c_2 is a *descendant* of c_1 if $V_1 \subset V_2$. c_1 is the *parent* of c_2 and c_2 is a *child* of c_1 if $V_1 \subset V_2$ and $|V_1| = |V_2| - 1$. These relationships also extend to cuboids and form a structure called the *cuboid lattice*. An example is shown in Figure 1. The "All" or *apex* cuboid holds a single cell where all its values among $(a_1, a_2, \ldots, a_n)$ are $*$. On the other extreme, the *base* cuboid at the bottom holds cells where none of its $(a_1, a_2, \ldots, a_n)$ values is $*$.

[1] For the sake of illustration, ignore the fact that the sample size is too small to be statistically meaningful.

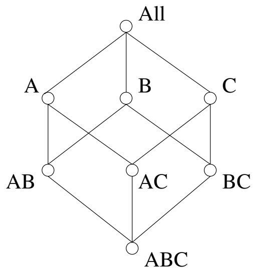


Figure 1: Cuboid lattice

Table 2 is an example of R. Each tuple corresponds to a person being sampled. The four attributes: `Gender`, `Age`, `Education`, and `Occupation`, segment the person. And finally, the value of the data is the `Income` of the person.

The query model follows the semantics of a data cube being constructed on the input relation R. More specifically, the user can pick any cell in C_R and ask for information about V for that cell. The measure of the cube (*e.g.*, average, sum) is computed on V and returned. In the motivating example, the first query of `Occupation`=*Programmer* is essentially a cell in the `Occupation` cuboid with all other dimensions set to $*$ or "don't care." The measure of the cube is the average of `Income`.

2.2 Problem Definition

As Example 3 showed, there are a few additional features than traditional OLAP. First, given α as a confidence level (*e.g.*, 95%), all cell queries to C_R should return a confidence interval of α confidence in addition to the measure as an answer. In comparison to traditional OLAP, the confidence level indicates the reliability of the measure to the user. Second, given *minsup* as a minimum support on the number of samples, if a cell's sample size does not satisfy *minsup* (*i.e.*, sample size $<$ *minsup*), "similar" cells in the data cube will be used to "boost" the confidence interval if certain criteria are satisfied. Lastly, if there is insufficient space to materialize the full data cube, a new methodology is needed to answer queries.

2.3 Confidence Interval

To make future discussions easier, a quick review of confidence interval calculation is given here [13].

Let x be a set of samples. The mean of the samples is denoted by $\bar{x}$, and the number of samples in x is denoted by l. Assuming that the standard deviation of the population is unknown, the *sample* standard deviation of x is denoted by s. Given some desired confidence level, the confidence interval for $\bar{x}$ is

$$\bar{x} \pm t_c \hat{\sigma}_{\bar{x}} \tag{1}$$

where t_c is the critical t-value associated with the confidence level and $\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$ is the estimated standard error of the mean. To find the appropriate t_c, one has to know the desired confidence level (*e.g.*, 95%) and also the degree of freedom, which is just $l - 1$.

3. THE SAMPLING CUBE FRAMEWORK

This section will describe the full framework of providing OLAP to sampling data. Specifically, it will describe how to store the sample data in a data cube structure named **Sampling Cube**, how to efficiently aggregate the data, and how to use similar neighboring cells to boost confidence. For now, only *full materialization* of this sampling cube is discussed. In other words, all cuboids and cells are constructed. The next section will discuss alternatives when full materialization is unrealizable.

3.1 Materializing the Sampling Cube

Given the base relation (base cuboid) in R, it is straightforward to construct C_R. There are many algorithms to efficiently do this [3, 22]. They all essentially provide an efficient way of traversing the cuboid lattice space in order to minimize the scanning of the data. In the sampling cube, the new question is *how to efficiently calculate the confidence interval at high-level cells.* The naïve way is to gather all the corresponding raw samples in the original input data and calculate the sample mean, sample standard deviation, and then the confidence interval. However, this is not very efficient since many cells contain a large number of samples. To repeat the computation for every cell in C_R is expensive and also redundant since the samples are shared between cells.

In traditional OLAP, this problem is solved by exploiting certain properties of the cube measure. There are two popular properties: **distributive** and **algebraic**. A measure is distributive if it can be computed solely based on the measures of its subsets, and a measure is algebraic if it can be computed based on a bounded number of measures of its subsets. Sum is an example of a distributive measure and mean is an example of an algebraic measure. These two properties are desirable because they facilitate very efficient aggregation. In this work, the measures of the cube are mean and confidence interval. Mean is known to be algebraic. The attention now turns to confidence interval. It is easy to see that it is definitely not distributive. But is it algebraic?

Lemma 1. *The confidence interval measure is algebraic.*

PROOF. There are three terms in the confidence interval computation. First is the mean of the sample set, $\bar{x}$. This was shown to be algebraic already. Second is the critical t-value, which is calculated by a lookup. With respect to x, it depends on l (count) and it is easy to see that count is distributive. The final term is $\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$, which also turns out to be algebraic if one records the linear sum ($\sum_{i=1}^{l} x_i$) and squared sum ($\sum_{i=1}^{l} x_i^2$). □

To summarize, the mean and confidence interval measures of the data cube are algebraic. At every cell, exactly three values are sufficient to calculate them; all of which are either distributive or algebraic. They are the following:

1. l
2. $sum = \sum_{i=1}^{l} x_i$
3. $squared\ sum = \sum_{i=1}^{l} x_i^2$

With this knowledge in hand, constructing the sampling cube is very clear now. Any of the previously developed cubing algorithms can be used with just aggregating the three above values in each cell. At query time, the three values are then used to compute the mean and confidence interval.

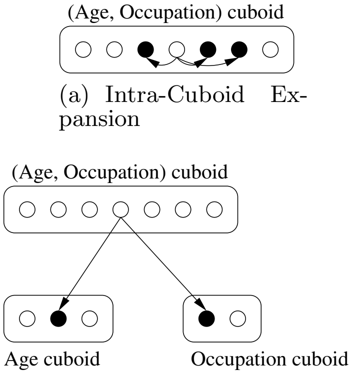
3.2 Boosting Confidence for Small Samples

Now that the sampling cube is materialized, the next step is to use it. Recall that queries are point or range queries posed against the cube. Without loss of generality, consider the case of a point query, which corresponds to a cell in C_R. The goal is to provide an accurate point estimate (in the example, the sample mean of `Income`) for the samples in that cell. Because the cube also reports the confidence interval associated with the sample mean, there is some measure of "reliability" to the answer. If the confidence interval is small, the reliability is deemed good. But if the interval is large, the reliability is questionable.

Consider what affects the size of the confidence interval. There are two main factors. The first is the variance of the sample data. A rather large variance in the cell may indicate that the chosen cube cell is not good for prediction and a better solution is probably to drill down on the query cell to a more specific one, *i.e.*, asking more specific queries. Second, a small sample size can cause a large confidence interval. When there are very few samples, the corresponding t_c is large because of the small degree of freedom. This in turn could cause a large confidence interval. Intuitively this makes sense. Suppose one is trying to figure out the average income of people in the United States. Just by asking 2 or 3 people does not give much confidence to the answer.

The best way to solve this small sample size problem is to simply get more data. This, however, is easier said than done. Gathering data is often the most expensive part of the analysis. So what can be done instead? Fortunately, there is an abundance of additional data available already. They do not match the query cell exactly, but they are conveniently organized in a structured data cube. Perhaps they can be used if certain criteria are met.

Figure 2 shows the two possible methods to "expand" the query and get more data to boost confidence. They both expand the original query in the data cube, just in different directions.



(a) Intra-Cuboid Expansion

(b) Inter-Cuboid Expansion

Figure 2: Query expansion within sampling cube

3.2.1 *Intra-Cuboid Query Expansion*

In the intra-cuboid case, the expansion occurs by looking at nearby cells in the same cuboid as the queried cell. But as mentioned before, careful consideration is needed before expansion. The new samples should only serve the purpose of increasing the confidence in the answer and *not* change the semantic of the query. There are two primary questions to answer. First, "Which dimension(s) should be allowed to expand?" And second, "Which value(s) within those dimension(s) should the expansion use?"

To answer the first question, dimensions which are *uncorrelated* or *weakly correlated* with the measure value (*i.e.*, the value to be predicted) are the best candidates for expansion. Expanding within these dimensions will likely increase the sample size and not shift the answer of the query. Consider an example of a 2D query specifying `Education` = "College" and `Birth Month` = "July". Let the cube measure be `average Income`. Intuitively, education has a high correlation to income while birth month does not. As a result, it would be harmful to expand the `Education` dimension to include values such as "Graduate" or "High School." They are likely to alter the final result. However, expansion in the `Birth Month` dimension to include other month values could be helpful, because it is unlikely to change the result but will increase sampling size.

To mathematically measure the correlation of a dimension to the cube value, the correlation between the dimension's values and their aggregated cube measures is computed. *Pearson's correlation coefficient* for numerical data and χ^2 value for categorical data are popularly used correlation measures (although there are many other measures, such as *covariance*, can be used) [13]. A dimension that is strongly correlated with the value to be predicted should *not* be a candidate for expansion. Notice that since the correlation of a dimension with the cube measure is independent of a particular query, it should be precomputed and stored with the cube measure to facilitate efficient online analysis.

Now that the possible dimension(s) for expansion have been selected, the next step is to select the values within those dimensions. This relies on the semantic knowledge of the dimensions in question. The goal should be to select semantically similar values in order to minimize the risk of altering the final result. Consider the `Age` dimension, similarity of values in this dimension is clear. There is a clear order to the values. For dimensions with *numerical* or *ranked* data (*e.g.*, `Education`), such an ordering is clear and one should select values closer to the instantiated query value. For categorical data with its dimension organized in a multilevel hierarchy in a data cube (such as location), one should select those values located in the same branch of the tree (such as in the same district or city).

When such domain knowledge exists, semantically similar cells maybe used to boost the confidence interval of the query cell. Figure 2(a) shows an illustration. But when such knowledge does not exist, one has to be very careful in how other cells are used. A naïve approach could be to simply compare the query cell vs. all other cells in the dimension and use the most similar. But this could easily fall into the trap of "self-fulfilling prophecy." This is a term used in sociology where pre-existing beliefs about a false outcome evoke behavior that actually brings the outcome to fruition. In the case of intra-cuboid expansion with no domain knowledge, one has to be careful of this problem. Just blindly using cells that contain similar values may bring about an answer that does not semantically meaningful.

Even though strongly correlated dimensions are precluded from expansion, yet another precaution should be taken to ensure that expansion does not alter the answer of the query. In other words, the new samples should share the same cube value (*e.g.*, mean income) as the existing samples in the query cell. A method in statistics to determine whether two samples have the same mean (or any other point estimate) is the **Two Sample T-Test** [13]. This is a relatively simple and statistically sound test used in many applications. Due to space restrictions, we will skip its definitions. At a high level, the test will determine whether two samples have the same mean (the null hypothesis) with the only assumption being that they are both normally distributed. The test fails if there is evidence that the two samples do not share the same mean. Furthermore, the tests can be performed with a confidence level as an input. This allows the user to control how strict or loose the query expansion will be. As it turns out, the aforementioned three values recorded at each data cube cell are sufficient to perform the two sample t-test. This allows the test to be performed efficiently given any two cells in the data cube.

Example 4 (INTRA-CUBOID EXPANSION). *Given the input relation in Table 2, let a query be "`Age` = 25" at 95% confidence. This returns an `Income` of $50,000 with a rather large confidence interval*[2]*. Since this confidence interval is larger than the preset threshold and the `Age` dimension was found to have little correlation with `Income` in this dataset, intra-cuboid expansion starts within the `Age` dimension. The nearest cell is "`Age` = 23," which returns an `Income` of $85,000. The two sample t-test at 95% confidence passes so the query expands; it is now "`Age` = {23, 25}" with a smaller confidence interval than initially. However, it is still larger than the threshold so expansion continues to the next nearest cell: "`Age` = 28", which returns an `Income` of $250,000. The two sample t-test between this cell and the original query cell fails; as a result, it is ignored. Next, "`Age` = 31" is checked and it passes the test. The confidence interval of the three cells combined is now below the threshold and the expansion finishes at "`Age` = {23, 25, 31}."*

In summary, dimensions not correlated with the cube measure are chosen as candidates for intra-cuboid expansion. Semantic similarity of the dimension's values are used to slowly expand a neighborhood of candidate cells around the query cell. For each candidate cell, the two sample t-test is performed to decide whether the candidate cell should be included in the expansion. When there is no semantic knowledge available, it might be unwise to expand unless the particular application calls for it.

3.2.2 Inter-Cuboid Query Expansion

The choices in inter-cuboid query expansion are slightly easier. Figure 2(b) shows an illustration. The expansion occurs by looking to a more general cell (drawn in black). In the figure, the cell in cuboid `Age, Occupation` can either use its parent in `Age` or `Occupation`. One can think of inter-cuboid as just an extreme case of intra-cuboid where *all* the cells within a dimension are used in the expansion. This essentially sets the dimension to $*$ and thus generalizes to a higher level cuboid.

Given a k-dimensional cell, there are k possible direct parents in the cuboid lattice. Though there are *many* more ancestor cells in the data cube if multiple dimensions are allowed to be rolled up simultaneously, only one is allowed here to make the search space tractable and also to limit the change in the semantics of the query. Using similar tests as the last section, correlated dimensions are not allowed in inter-cuboid expansions. Within the uncorrelated dimensions, the two sample t-tests can be performed to confirm that the parent and the query cell share the same sample mean. If multiple parent cells pass the test, the confidence level of the test can be adjusted progressively higher until only one passes. Alternatively, multiple parent cells can be used to boost the confidence simultaneously. The choice is application dependent.

Example 5 (INTER-CUBOID EXPANSION). *Given the input relation in Table 2, let the query be "`Occupation` = Teacher $\wedge$ `Gender` = Male." There was only one matching sample in Table 2 with `Income` = $80,000. Suppose the corresponding confidence interval is larger than the preset threshold. There are two parent cells in the data cube: "`Gender` = Male" and "`Occupation` = Teacher." By moving up to "`Gender` = Male" (and thus setting `Occupation` to $*$), the mean `Income` is $101,000. A two sample t-test reveals that this parent's sample mean is not the same as the original query cell's. So it is ignored. Next, "`Occupation` = Teacher" is tried. It contains a mean `Income` of $85,000 and passes the two sample t-test. As a result, this new value is used and the query is expanded to "`Occupation` = Teacher."*

Though the above method will work for inter-cuboid expansion, there is another method which makes more sense computationally. Instead of looking at the problem as expanding from the query cell *up* to a more general cell, one could look at it as the more general cell looking *down* at the query cells (*i.e.*, its children) and making its determinations about expansion. This method leads directly into the next section about the sampling cube shell.

3.2.3 Expansion Method Selection

Before moving on, some discussion is needed about intra-cuboid expansion vs. inter-cuboid expansion. This is a difficult question to answer a priori without knowing the data and the application. The first guideline in choosing between the two should be what is the tolerance for change in the semantics of the query. This depends on the specific dimensions chosen in the query. For instance, the user might tolerate a bigger change in semantics for the `Age` dimension than `Education`. The difference in tolerance could be so large that he/she is willing to set `Age` to $*$ (*i.e.*, inter-cuboid expansion) than letting `Education` change at all.

If no domain knowledge is available, the main quantitative guides are the correlation coefficients and the two sample t-test between the query cell and the possible expansion cells. The value of the correlation coefficient is an indication of expansion's safety. And by progressively setting higher confidence levels in the test, one could choose between the two expansion methods by seeing which one passes the higher confidence level test. This offers a numerical way of comparing between the two choices, but in a real world application, domain knowledge is definitely a better method of making the ultimate choice.

4. THE SAMPLING CUBE SHELL

So far, the discussion has only focused on the full materialization of the sampling cube. In many real world problems,

[2] For the sake of the example, suppose this is true even though there is only one sample. In practice, there should be a few more points to calculate a legitimate value.

this is often impossible. Even given a modest number of dimensions in the base data, constructing the whole data cube can be prohibitive. Recall that the number of cuboids in a data cube is exponential with respect to the number of dimensions. So even with just 20 dimensions, 2^{20} cuboids can be quite a pain to handle. The real world survey data used in this work's experiments contains over 600 dimensions!

Clearly, another methodology is needed. Ideally, this new methodology should be able to provide the same or close to the same answers as the full sampling cube with a much smaller computation and storage requirement.

To motivate the proposal, first imagine what a typical query will be. An analyst will select some specific values in some dimensions and ask for the confidence interval in that cell. But most likely, the number of dimensions specified in the query will be low (≤ 5: *e.g.*, `Age`, `Gender`, `Marital Status`, `Income Level`, etc.). To specify a high dimensional cell is to target a very specific segment of the population. One that is probably too specific to be of any value. This means that high dimensional cuboids are probably not needed.

Second, consider what would happen if `Birth Month` were a dimension in Table 2. Clearly, there should not be any correlation between the month of a person's birth date and his or her income level. This can be statistically verified by checking the standard deviation or the confidence level (both of which should be large) of the cells in the `Birth Month` cuboid. Now recall the ultimate goal of the user. It is to extract meaningful information about the cube value (*e.g.*, `Income`). If the sample standard deviation of the value is high for many cells in a cuboid, it indicates that there is little information to be found in this cuboid. Therefore, there is probably little utility in presenting the cuboid to the user. Furthermore, additional higher level cuboids that combine with `Birth Month` can probably be skipped, too. This drastically cuts down on the size of the sampling cube since it essentially removes one dimension.

This motivation leads directly to the proposal of the **Sampling Cube Shell**. As the name suggests, it is a "shell" around the complete sampling cube that only computes some of the outer layers. Figure 3 shows a sample illustration. In it, only a portion of the cuboids are materialized (the shaded ones). They are the shell around the data cube and will be used to answer the queries.

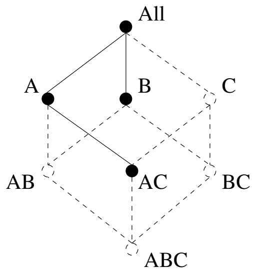


Figure 3: Sampling Cube Shell

4.1 Building the Sampling Cube Shell

The algorithm to build the sampling cube shell is top-down. It starts at the apex cuboid and proceeds down the cuboid lattice towards the base cuboid. The search in this space is iterative and greedy: in each iteration, the best candidate cuboid is chosen and added to the shell. This process halts until some stopping condition is met. The condition could be a space constraint, *i.e.*, number of cuboids built cannot exceed some value. Or it could be an information constraint, *i.e.*, the gain in building a new cuboid must exceed some minimum.

4.1.1 Cuboid Standard Deviation

The first question is how to compare cuboids (in order to get to the greedy selection). This requires a measure on the "goodness" of a cuboid.

Definition 1 (Cuboid Standard Deviation). *Given a cuboid B with m cells $\{c_1, \ldots, c_m\}$, the* Cuboid Standard Deviation *(CSD) is defined as*

$$CSD(B) = \frac{\sum_{i=1}^{m} s(c_i) \times \frac{n(c_i)}{n(B)}}{1 - \frac{\sum_{i=1}^{m} small(c_i)}{m}}$$

where $s(c_i)$ is the sample standard deviation of the samples in c_i, $n(c_i)$ is the number of samples in c_i, $n(B)$ is the total number of samples in B, and $small(c_i)$ is a function which returns 1 if $n(c_i) \geq minsup$ and 0 otherwise. If the denominator is 0, CSD is defined to be ∞.

The CSD of a cuboid measures the amount of variance with respect to the cube value in its cells. Obviously, the lower the value of CSD, the better the cuboid is at capturing the correlation between its cells and the value. The definition of CSD achieves this using a linear combination of its cells' standard deviations. In the final summation, the standard deviations are weighted by the sizes of the cells. As a result, if only a small percentage of the cells have low standard deviation but they hold the majority of the sampled data, they will have a large effect on the CSD of the cuboid.

For practical reasons, two minor adjustments are made to the CSD calculation. First, if $s(c_i)$ is very small ($< minsd$), it is set to 0. In other words, if the standard deviation of the cube values in a cell is already quite small, it is unnecessary to further examine that cell's subsets since they would contain the same behavior. Setting the $s(c_i)$ to 0 reflects this notion. Second, if $n(c_i) < minsup$, c_i is ignored. In such cases with so few samples in the cell, it is meaningless to measure any information from them. But, a situation could arise where a large portion of the cuboid's cells have small $n(c_i)$. For example, consider a dimension storing unique IDs of samples. In this case, the standard deviation of all cells would be 0 since each cell contains exactly one sample. The CSD is low at 0, but the cuboid is actually useless since its cells do not offer any generalization power. To penalize this type of situation, the denominator in the CSD calculation is set to reweigh the standard deviation calculations in the numerator by the percentage of so called "small" cells in the cuboid. Both these adjustments will come in handy later on during the construction of the cube shell.

4.1.2 Cuboid Standard Deviation Reduction

Given CSD as a measure of a cuboid's cells' correlation with the cube value, it is now possible to compare different cuboids quantitatively. However, in order to use it in the construction algorithm, another definition is needed to measure the *incremental* gain of a cuboid.

Definition 2 (Cuboid Standard Deviation Reduction). *Given a cuboid B and $parents(B)$ as the set of B's parents* *in the cuboid lattice, the* Cuboid Standard Deviation Reduction *(CSDR) is defined as*

$$CSDR(B) = \left[\min_{B' \in \, parents(B)} CSD(B') \right] - CSD(B)$$

The CSDR of a cuboid measures the *reduction* in CSD from one of its parents. Because the data cube is a lattice and not a tree, a cuboid can have multiple parents. To maximize the gain, the reduction is measured from the best parent.

4.1.3 Cube Shell Construction

Building the sampling cube shell is a top-down and greedy process. It uses CSDR to select the best cuboid in each step of growing the cube shell. Initially, only the *All* or apex cuboid exists. By definition, it contains exactly one cell and the standard deviation of it is the standard deviation of all the samples put together. The child cuboids of the apex cuboid are then added into a *candidate set*. The CSDR of each cuboid is computed. The candidate cuboid with the best CSDR is chosen and added to the shell. Its children in the cuboid lattice are added to the candidate set. This process iterates until a stopping criterion is met. Note that the final data structure is not strictly a tree since a node could have multiple parents. It is just a portion (*i.e.*, shell) of the complete data cube lattice.

Two pruning methods are used in order to improve both the efficiency and the effectiveness of the resultant shell. They are directly related to the two adjustments made to the CSD calculation earlier.

First, if a cell's standard deviation is very low ($< minsd$), its descendant cells are removed from future consideration. The reason for this is that if the samples in a cell already share basically the same value for the point estimate, it is pointless to examine its sub-segments since most likely they will just produce the same values. This, in effect, achieves inter-cuboid query expansion. At runtime, if the query cell is one of the pruned descendant cells, it will not be found but the parent cell will be. The point estimate from the parent cell will then be used in substitute, but it will be fairly accurate since the standard deviation in its samples is so low. In essence, the query cell has been expanded to the parent cell a priori. Detailed discussion of query processing will be given in the next section.

Second, for every cell, if its sample size does not satisfy a minimum support threshold ($< minsup$), its descendant cells in the descendant cuboids are removed from future consideration. Intuitively, this supports the idea that if a segment is already very small, it is fruitless in analyzing its sub-segments, which could only get smaller. This is essentially the idea of the iceberg cube [3].

Algorithm 1 shows the shell construction algorithm in pseudo-code.

Algorithm 1 (Cube Shell).

Input: *(1) Input table R; (2) minsup; (3) minsd*

Output: *Sampling cube shell S*

Method:

```
Candidates = {apex cuboid of R}
while Candidates $\neq \emptyset$ or halting criteria not met
    B = cuboid in Candidates with largest CSDR
    remove B from Candidates
    add B to S
    add B's descendant cuboids to Candidates
    update CSD values in Candidates
return S
```

There are two possible ways the algorithm could halt, and the choice between them depends on the application. The first is the size of the shell. This is a natural halting criterion if storage space is the primary concern. As the next section and also later experiments will show, the bigger the shell, the higher the quality of query processing. The second criterion could be a minimum CSDR. At each iteration, if the largest CSDR in the candidate set does not exceed some minimum, the algorithm halts.

Example 6 (Shell Construction). *To make the algorithm more concrete, consider how it will work on the example data in Table 2. Initially, the candidate set only includes the apex cuboid of R. The CSD of this cuboid is simply the standard deviation of all rows in R, which is 143,760. Next, since it is the only cuboid in the candidate set, it is added to the shell. Its four descendants, which are the one-dimensional cuboids,* `Gender`*,* `Age`*,* `Education`*, and* `Occupation` *are added to the candidate set. Table 4(a) shows the CSD and CSDR of each candidate cuboid. For the sake of this example, values in the* `Age` *cuboid are binned into 4 ranges: 21–30, 31–40, 41–50, and 51+. This produces a more reasonable CSD value for the* `Age` *cuboid. Since these cuboids all share the same parent, their CSDR are all calculated with respect to the apex. This ends the first iteration of the algorithm.*

In the second iteration, the best candidate cuboid according to its CSDR value in Table 4(a), the `Occupation` *cuboid, is added to the shell. Its descendants, which are all the 2D cuboids that extend from* `Occupation`*, are added to the candidate set. Their CSDR values are calculated with respect to* `Occupation`*. Figure 4(b) shows the structure of the shell after the addition, and Table 4(c) shows the new candidate set with CSD and CSDR values.*

In the next iteration, the cuboid with the best CSDR, `Age`*, is added to the shell. Figure 4(d) shows the result of this addition. Its descendants are added to the candidate set and the algorithm continues.*

4.2 Query Processing

The final step is to process queries using the sampling cube shell. Recall that the query model consists of point queries in the data cube of the input relation. Because the sampling cube shell is not the complete data cube and does not contain all possible cuboids, there are three possibilities at query time.

4.2.1 Exact Dimension Match

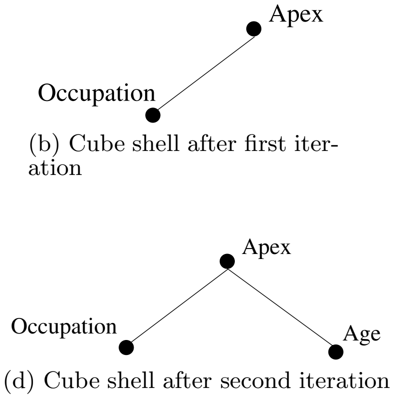
First, the queried dimensions match one of the cuboids in the shell. In this case, the answer to the query exists in the shell already and it is simply returned.

4.2.2 Subset Dimension Match

Second, the queried dimensions are a subset of one of the cuboids in the shell. For instance, the queried dimension could be `Gender` and only the `(Age, Gender)` cuboid (and not the `Gender` cuboid) exists in the shell. This is entirely possible due to the build order of the construction algorithm. In this case, the exact answer is produced by scanning the superset cuboid in the shell for the appropriate rows and computing the necessary values on the fly.

Candidate Cuboid	CSD	CSDR
`Gender`	139,196	4,564
`Age`	102,836	40,924
`Education`	109,485	34,275
`Occupation`	43,852	**99,908**

(a) Candidate set after first iteration



(b) Cube shell after first iteration

(d) Cube shell after second iteration

Candidate Cuboid	CSD	CSDR
`Gender`	139,196	4,564
`Age`	102,836	**40,924**
`Education`	109,485	34,275
`(Gender, Occupation)`	13,338	30,514
`(Age, Occupation)`	45,287	-1,435
`(Education, Occupation)`	6,261	37,591

(c) Candidate set after second iteration

Figure 4: Sampling cube shell construction example

4.2.3 Superset Dimension Match

So far, the answers produced have been lossless with respect to the full sampling cube. They either exist in the shell or can be computed in the shell from more specific cells. The last case to handle is when the queried dimensions are a *superset* of all cuboids in the shell. Though technically the query model allows any cell in the data cube to be queried, most likely it will be in the low dimensional ones (≤5 dimensions). This was one of the motivations of the sampling cube shell in the first place. In other words, if this case occurs, most likely the sampling cube shell will contain a cuboid that is not very far off from the queried dimensions. For example, the queried dimensions could be `(Age, Gender, Occupation)`, and the largest cuboid in the shell only contains two dimensions.

In this case, a careful assessment is needed. In general, the queried dimensions could have a superset relationship to *multiple* cuboids in the shell. In the example above, both `(Age, Gender)` and `(Age, Occupation)` could exist in the shell and be used to answer the query. This raises two questions. First, which cuboid in the shell should be used? And second, how will the cuboid be used to answer the query?

In general, let there be k cuboids, $B_1 \dots B_k$, in the sampling cube shell whose dimensions are subsets of the queried dimensions. Other than scanning base table, these k cuboids are the only sources of information about the query cell in the shell. But since they are all more general than the query cell, the final answer will have to be approximated from them.

The question is then which of the k cuboids should be used. The first goal should be to pick the cuboid that is closest to the query cuboid in the cuboid lattice. Semantically, this is the closest source of information. In general, multiple cuboids could tie for being the closest. Let there be k_0 of these where $k_0 \leq k$. Within these k_0 cuboids, the *average* of the point estimates at the relevant cells is then the point estimate answer for the query. In testing, several methods were tried, including weighted average by sampling size, weighted average by confidence interval size, choosing the cuboid with the highest confidence, and choosing the cuboid with the smallest sampling size. The simple average turns out to be the best due to the fact that it is not affected by any biases in sampling which could place an uneven number of samples in different cuboid cells. In testing, this was also confirmed to be the best on average.

Figure 5 shows a sample query in the `(Age, Occupation)` cuboid. Suppose that cuboid does not exist in the cube shell, but `Age`, `Occupation`, and the apex do. As a result, there are 3 ancestor cells in the cube shell. The cells in `Age` and `Occupation` are 1 hop away from the query cell in the cuboid lattice and the apex cell is 2 hops away. `Age` and `Occupation` tie for being closest to the query cell; the apex is ignored. The average of the `Age` and `Occupation` cells is then the answer to the `(Age, Occupation)` query.

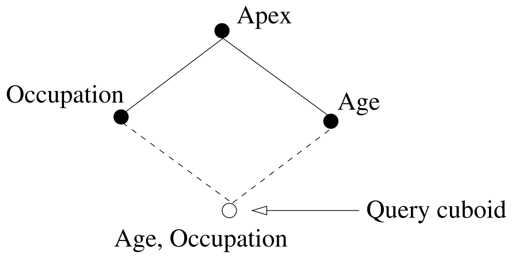


Figure 5: `(Age, Occupation)` query

5. PERFORMANCE EVALUATIONS

This section shows various evaluations of the sampling cube shell with real world data. Everything is implemented in C++ and compiled with GCC. Experiments were performed on a Linux machine with an Intel Pentium4 2.4GHz CPU and 2GB of memory.

Real sampling data from a Fortune 100 company was obtained for the tests. For confidentiality reasons, the name of the company, the names of products, or actual values cannot be revealed. The data contains over 750,000 samples and nearly 600 dimensions. Each sample is a record of a sale of a particular product to a single customer. The customer is then surveyed on various dimensions such as age, marital status, employment status, education level, etc.

Two subsets are extracted from the full data. One is a 21-dimensional dataset with the number of children (under age 16) as the cube value. The other is a 22-dimensional dataset with the household income as the cube value. In the original input data, income was already binned into 12 ranges. In testing, a random income value within the range is generated. As a result, some of the subtle patterns might have been lost but the big correlations should still hold.

In all tests, *shell_size* indicates the size of the sampling cube shell (*i.e.*, number of cuboids). It is the halting criterion for shell construction. Unless mentioned differently, *minsup* and *minsd* are both set to 0.

5.1 Shell Construction Efficiency

As mentioned previously, materializing the full sampling cube is often unrealistic in real world scenarios. It is known to be exponential in the number of dimensions. But what about the sampling cube shell? Figure 6 shows the time to compute cube shells of *shell_size* 20 and 50 as dimensionality increases from 5 to 20. For comparison, the full data cube (as computed by BUC [3]) is also shown. As expected, BUC explodes when the number of dimensions reaches higher than 15. In comparison, the cube shells grow linearly. This is not surprising because the time is constrained by *shell_size*. The reason the time does increase; however, is because the number of candidates to be examined increases as the number of dimensions does. For example, with the number of dimensions at 19 and *shell_size* at 50, the total number of candidates generated is 778. This is far smaller than the full cube (2^{15}) but it is also much larger than *shell_size*.

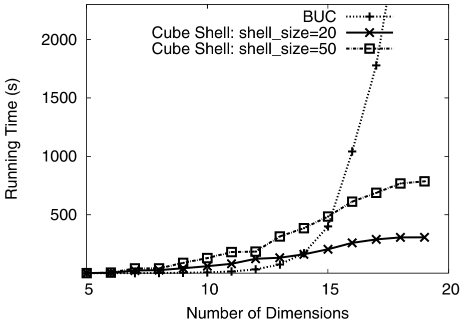


Figure 6: Materialization time vs. dimensionality

Next, efficiency with respect to the number of tuples in the input relation is checked. In traditional OLAP, this usually has a linear relationship to running time due to a linear increase in the number of overall cells. Figure 7 shows that this is also the case with the sampling cube shell. As the number of tuples increases from 20,000 to 100,000 in a 22 dimensional data set, the time to compute a cube shell (regardless of *shell_size*) is linear to the size of the input.

5.2 Query Effectiveness

Experiments in this section will address the two major claims on the effectiveness of the proposed framework. First, they will show that query expansion increases the reliability of the query answer. Second, they will show that query processing with the sampling cube shell produces negligible errors while reducing the space requirement significantly.

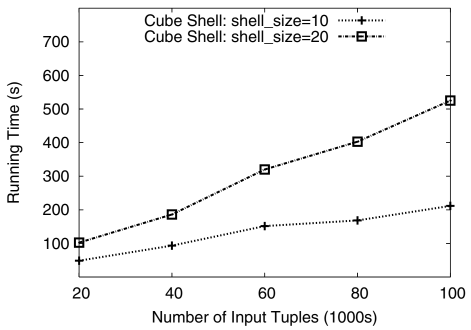


Figure 7: Materialization time vs. number of tuples

5.2.1 Query Expansion

To test the effectiveness of query expansion, the full input data of 750,000 tuples is taken to be the population and a random sample is taken from it. A full sampling cube is constructed on the sample, and query results are compared to the population in order to measure the accuracy.

In the first experiment, a sampling cube measuring the `average Number of Children` is built from a random 0.1% sample of the full input. Three dimensional point queries consisting of the `Gender`, `Marital`, and `Age` dimensions are given to the (1) full data, (2) sampling cube *without* query expansion, and (3) sampling cube *with* query expansion. The age dimension is allowed to expand to at most ± 2 years of the query age. The output below shows a sample query for `Gender` = *Female*, `Marital` = *Married* and `Age` = *36*.

```
> q GENDER FEMALE MARITAL MARRIED AGE 36
Population:           1.66 from 4783 points
Sample w/o expansion: 2.33 +/- 1.12 from 6 samples
Sample w/ expansion:  1.51 +/- 0.34 from 47 samples

  Difference in mean w/o expansion: 0.67
  Difference in mean w/ expansion:  0.15
```

As the output shows, 1.66 is the "correct" answer from (1). 2.33 ± 1.12 is the answer from (2) and 1.51 ± 0.34 is the answer from (3). Here, query expansion results in a significant improvement in the query answer. 1.51 is much closer to 1.66 than 2.33. The sample size also increases from 6 to 47, which reduces the 95% confidence interval.

Table 3(a) shows the full effect of intra-cuboid expansion in the `Age` dimension over many queries, similar to the above example. The first two columns of each line show the query values of the `Gender` and `Marital` dimensions. In each line, the `Age` dimension is enumerated over all possible values (approximately 80 distinct values), and each combination forms a distinct 3 dimensional point query in the sampling cube. The third and fourth columns show the average absolute error in the query results of the sampling cube without and with query expansion. As the fourth and fifth columns show, turning on intra-cuboid query expansion in the `Age` dimension improves the accuracy significantly. The last row in the table shows an average of 26% improvement from nearly 500 different queries. The last three columns in the table show the effect of query expansion on sampling size. Without expansion, the number of samples per query is only 1.4. With expansion, it increases to 13.4.

Table 3(b) shows a similar experiment with the `Age` dimension and the average Household Income as the cube measure. In this experiment, 0.05% of the input data is loaded into the sampling cube, and the age dimension is again allowed to expand ± 2 years from the query age. The two dimensions queried in addition to Age are Gender and Education. The result of this experiment is similar to the last one. In the nearly 650 queries executed, the average error reduction from no expansion to intra-cuboid expansion is 51%. Average sampling size also increases significantly.

Lastly, Table 3(c) shows another experiment. In this one, the average Household Income is still the cube measure. But the expansion is now within the Number of Children dimension, which is allowed to expand to ± 1 child. Three other dimensions were specified in the query, namely Gender, Marital, and Education. Due to limited space, only the final average is shown in Table 3(c). Again, both query accuracy and sampling size are improved. The experiment also shows the average reduction in the size of the confidence interval as a result of query expansion. With more sampling points, it is easy to see why the interval size would decrease. This, in addition to the reduction in the mean's error, improves the overall quality of the query results.

5.2.2 Sampling Cube Shell

Next, the query accuracy of the sampling cube shell is measured. The same query is given to the sampling cube shell and the full sampling cube, and the difference between the answers is the "error." The full sampling cube is simulated by scanning the input relation repeatedly since it was not possible to full materialize it.

First, a sanity check is performed by looking at the cuboids chosen by the cube shell according to CSD and CSDR. In the dataset measuring the number of children, the Age, Age Cohort, and Generation cuboids are chosen as the first three. Intuitively, these checkout to be sensible choices.

Figure 8 show the effect of *shell_size* on query accuracy using the dataset of average household income. 1000 random queries ranging anywhere from 1D to 5D were processed both by the sampling cube and the sampling cube shell. The absolute percent error is shown, which is defined as (|query answer from the shell cube − query answer from full cube|) ÷ query answer from the full cube.

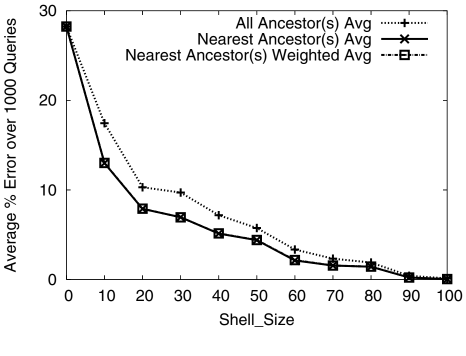


Figure 8: Query accuracy vs. *shell_size* for average household income dataset

In Figure 8, there are three curves. They show different methods of calculating the superset queries mentioned in Section 4.2.3. In "All Ancestor(s) Avg," the answer at the query cell is computed by taking the average of *all* ancestors in the cuboid lattice. In "Nearest Ancestor(s) Avg," the answer is computed by taking the average of the nearest ancestors in the cuboid lattice. And lastly, in "Weighted Nearest Ancestor(s) Avg," the answer is computed by taking the average of the nearest ancestors inversely weighted by the size of the sampling set. As all three curves show, as *shell_size* increases, the error decreases. This is expected because there are more cuboids in the shell to accurately answer queries. Amongst the three curves, "All Ancestor(s)" gives the largest amount of error while "Nearest Ancestor(s) Avg" gives the least. "Weighted Nearest Ancestor(s) Avg" is close but just slightly worse. The reason is that most of the time, there was only one nearest ancestor; thus the method of averaging only had a small effect.

Clearly, there is a tradeoff between storage space requirement and query processing quality. But considering that the full sampling cube size contains 2^{22} cuboids, a sampling cube shell with *shell_size* set to 100 uses less than 0.01% of the space required while producing less than 1% error. Even if one does not compute the full sampling cube and materializes only 5D or smaller cuboids, a sampling cube shell with *shell_size* set to 100 still uses less than 1% (100/35,442) of the space required.

Figure 9 shows an experiment testing the effect of a query's dimensionality on the error of the cube shell. The average household income dataset is used with *shell_size* fixed at 50. As expected, as the query drills further down into the data cube, the error increases. This is because the query cells are further away from the boundaries of the cube shell. As a result, the error in the parent cells' approximation of the answer increases. But most OLAP queries will not drill down very far. In most real world situations, the analyst will only specify 1D to 5D queries. In these cases, the average error is very low (< 5%) while using less than 0.01% of the space required by the full sampling cube.

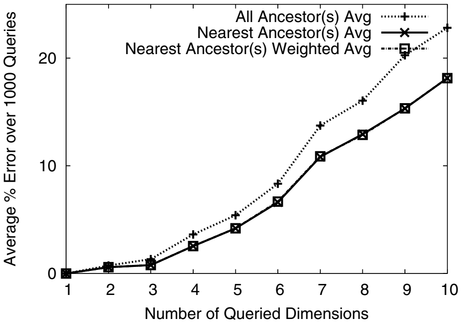


Figure 9: Query accuracy vs. query dimensionality for average household income dataset

6. RELATED WORK

Many algorithms have been proposed in traditional data cubes [10, 3, 22]. In comparison to this work, they only focus on full materialization and do not address the needs of sampling data. Selective materialization of the data cube has been studied before [12], but the criteria for selection was to simply reduce I/O in answering simple aggregation queries. Compression of data cubes [14, 15, 20], either lossy or lossless, also address the issue of selective computation. But again, the goal is to reduce I/O or storage space. The

Table 3: Query expansion experimental results

(a) Intra-Cuboid Expansion with `Age` dimension and `Average Number of Children` cube measure

Query		Average Query Answer Error			Sampling Sizes		
Gender	Marital	No Expand	Expand	% Improve	Population	Sample	Expanded
FEMALE	MARRIED	0.48	0.32	33%	2473.0	2.2	28.3
FEMALE	SINGLE	0.31	0.21	30%	612.6	0.6	6.4
FEMALE	DIVORCED	0.49	0.43	11%	321.1	0.3	3.4
MALE	MARRIED	0.42	0.21	49%	4296.8	4.4	37.6
MALE	SINGLE	0.26	0.21	16%	571.8	0.5	3.6
MALE	DIVORCED	0.33	0.27	19%	224.7	0.2	1.2
Average		0.38	0.27	26%	1416.7	1.4	13.4

(b) Intra-Cuboid Expansion with `Age` dimension and `Average Household Income` cube measure

Query		Average Query Answer Error			Sampling Sizes		
Gender	Education	No Expand	Expand	% Improve	Population	Sample	Expanded
FEMALE	HIGH SCHOOL	$55622	$31580	43%	641.4	0.3	3.9
FEMALE	SOME COLLEGE	$61526	$28822	53%	980.0	0.5	4.5
FEMALE	COLLEGE	$73309	$14504	80%	1132.8	0.5	6.8
FEMALE	POSTGRADUATE	$88658	$57907	34%	689.6	0.3	2.2
MALE	HIGH SCHOOL	$55671	$23503	57%	857.0	0.4	2.6
MALE	SOME COLLEGE	$63821	$34944	45%	1219.0	0.6	4.4
MALE	COLLEGE	$71120	$28913	59%	1511.0	0.9	7.4
MALE	POSTGRADUATE	$103619	$61191	40%	1191.8	0.6	4.2
Average		$71668	$35170	51%	1027.8	0.5	4.5

(c) Intra-Cuboid Expansion with `Number of Children` dimension and `Average Household Income` cube measure

	Average Query Answer Error				Sampling Sizes		
	No Expand	Expand	% Improve	CI Reduce	Population	Sample	Expanded
Average	$60873	$32458	51%	18%	713.9	3.6	12.3

question of how selectivity relates to confidence intervals is not addressed. Further, the question of how to handle sparse cells is not studied either. For high dimensional data, the shell fragment data structure was proposed [16] to compute only portions of the data cube. But compared to this work, there is no criterion for choosing which cuboids to build.

More recently, the prediction cube [7], was proposed. It integrates OLAP with machine learning by placing a learning model in each data cube cell. An optimal subspace, with respect to classification accuracy, is then discovered through OLAP operations. In comparison to this work, the prediction cube requires full materialization of the data cube, which is prohibitive in high dimensional data. Further, the prediction cube does not address the problem of sparse cells when there is no enough data available for effective learning.

Regression trees and decision trees [17, 21] are algorithms that are similar to the sampling cube shell. Both aim to model a value (either a class label or a numerical value) with a tree-like data structure. However, in regression trees and decision trees, nodes *split* the data into disjoint sets. The union of all leaf nodes in the final tree is the input data, and the intersection of all leaf nodes is empty. In the sampling cube however, nodes do not split data; they are simply *projections* of the same data into different subspaces. As a result, the criteria for node selection and the data in the nodes are very different.

Related to the modeling trees are feature selection algorithms [11]. Given either labeled or unlabeled data, such algorithms can find the most effective features (dimensions) for the particular problem, *e.g.*, classification, clustering. In the sampling cube, one could view the samples as being some kind of label, and the sampling cube shell essentially chooses the best dimensions. However, because of the OLAP space, the combinations of features are uneven. In feature selection, one simply chooses the m best features out of n where $m < n$. In the sampling cube shell, *different combinations* within the n features has to be considered because the query can be in any subspace. As a result, the search space is substantially larger than simple feature selection.

One area of study where subspace projections are specifically considered is subspace clustering or outlier detection [18, 1]. The goal is to find clusters or outliers in arbitrary subspaces of a high-dimensional space. This is a very different goal from the sampling cube shell. In sampling data, the distance between two data points, with respect to their dimension values, is irrelevant. If the dimensions are categorical, there might not even be distance functions defined. Rather, it is the "distance" between the *samples* held by the points that is at question.

Querying over uncertain data is another related field. In the OLAP context, there has been work [4, 5] that address the problem of uncertain values in the input data. The model generates multiple "worlds" with different probabilities, and the query essentially returns a combination of all possible worlds. This kind of uncertainty is very different from the case in the sampling cube. Sampling data's uncertainty is inherent in the data even though the collection might be 100% correct. In the relational data context, uncertainty has also been addressed by extending the relational database [2]. The lineage (where the data came from) and the uncertainty (possibly different values in the data) are incorporated into a relational system. Again, these uncertainty issues are different than the ones found in sampling data. In relational data, an attribute value has uncertainty in the sense that it could take on several different values. In sampling data, the uncertainty comes from that the value might not represent the full population, which is unseen. These are two very different uncertainties.

Query expansion in information retrieval is a major area related to the sampling cube's expansion of cube queries. In the context of information retrieval, query expansion expands the set of query terms in order to retrieve better documents [9, 8]. The underlying assumption is that the user is not aware of the optimal query terms and needs some assistance. The same reason exists in the sampling cube but the execution is very different. In information retrieval, the set of documents is view as *one* flat set. In sampling cube, there is a multidimensional structure to *many* different sets. This leads to a very different expansion process.

And finally, data cleaning is another related field. When attribute values in the input relation are missing or incorrect, various techniques can be used to correct them [19, 6]. This is somewhat like intra-cuboid query expansion in the sampling cube in the sense that other tuples are used to determine something about one particular tuple. But, the relationship between the tuples and sampling data is unique to the sampling cube.

7. REPEATABILITY ASSESSMENT RESULT

Figures 6, 7 and 8 have been verified by the SIGMOD repeatability committee.

8. CONCLUSIONS

In this paper we introduce a Sampling Cube framework along with an efficient Sample Cube Shell structure for analyzing multidimensional sampling data. Based on an OLAP framework, the sampling cube efficiently returns confidence intervals for multidimensional queries against the input data. One common problem in multidimensional data is that one can easily reach a cell with very few points even if the overall data size is large. This causes problems in analysis because small sample sizes results in confidence intervals that are too broad to be useful. The sampling cube exploits the structure of the dimensional space to expand the query when there is not enough data to answer the query confidently. Finally, the sampling cube shell solves the problem of high dimensional data. When there is a medium or large number of dimensions in the input, it becomes impossible to fully materialize the sampling cube since the size grows exponentially with the number of dimensions. To solve this problem, the shell only computes a subset of the full sampling cube. The subset consists of relatively low dimensional cuboids (common to be queried) and also cuboids that offer the most benefit to the user. In tests with both real world and synthetic data, the proposed framework is both efficient and effective in answering queries.

Extensions of the Sampling Cube framework with uncertainty reasoning, data cleaning, and data integration could be interesting themes in future research.

9. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *SIGMOD'01*.

[2] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB'06*.

[3] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD'99*.

[4] D. Burdick, P. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Olap over uncertain and imprecise data. In *VLDB'05*.

[5] D. Burdick, A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Olap over imprecise data with domain constraints. In *VLDB'07*.

[6] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD'03*.

[7] Bee-Chung Chen, Lei Chen, Yi Lin, and Raghu Ramakrishnan. Prediction cubes. In *VLDB'05*.

[8] P.-A. Chirita, C. S. Firan, and W. Nejdl. Personalized query expansion for the web. In *SIGIR'07*.

[9] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Query expansion by mining user logs. *IEEE TKDE'03*.

[10] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *ICDE'96*.

[11] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. In *Journal of Machine Learning Research*, 2003.

[12] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD'96*.

[13] W. L. Hays. *Statistics*. CBS College Publishing, New York, NY, 1981.

[14] L. V. S. Lakshmanan, J. Pei, and J. Han. Quotient cube: How to summarize the semantics of a data cube. In *VLDB'02*.

[15] L. V. S. Lakshmanan, J. Pei, and Y. Zhao. QC-Trees: An efficient summary structure for semantic OLAP. In *SIGMOD'03*.

[16] X. Li, J. Han, and H. Gonzalez. High-dimensional OLAP: A minimal cubing approach. In *VLDB'04*.

[17] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[18] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: A review. *SIGKDD Explorations*, 2004.

[19] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB'01*.

[20] Y. Sismanis and N. Roussopoulos. The complexity of fully materialized coalesced cubes. In *VLDB'04*.

[21] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann, 2005.

[22] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *VLDB'03*.