Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions

Jayavel Shanmugasundaram
Microsoft Research and
University of Wisconsin
Madison, WI 53706
jai@cs.wisc.edu

Usama Fayyad
Microsoft Research
One Microsoft Way
Redmond, WA 98052
fayyad@microsoft.com

P. S. Bradley
Microsoft Research
One Microsoft Way
Redmond, WA 98052
bradley@microsoft.com

ABSTRACT

Efficiently answering decision support queries is an important problem. Most of the work in this direction has been in the context of the data cube. Queries are efficiently answered by pre-computing large parts of the cube. Besides having large space requirements, such pre-computation requires that the hierarchy along each dimension be fixed (hence dimensions are categorical or pre-discretized). Queries that take advantage of pre-computation can thus only drill-down or roll-up along this fixed hierarchy. Another disadvantage of existing pre-computation techniques is that the target measure, along with the aggregation function of interest, is fixed for each cube. Queries over more than one target measure or using different aggregation functions, would require pre-computing larger data cubes. In this paper, we propose a new compressed representation of the data cube that (a) drastically reduces storage requirements, (b) does not require the discretization hierarchy along each query dimension to be fixed beforehand and (c) treats each dimension as a potential target measure and supports multiple aggregation functions without additional storage costs. The tradeoff is approximate, yet relatively accurate, answers to queries. We outline mechanisms to reduce the error in the approximation. Our performance evaluation indicates that our compression technique effectively addresses the limitations of existing approaches.

Keywords

OLAP, data cubes, clustering, density estimation, approximate query answering, data mining.

1. INTRODUCTION

There has been much work on answering multi-dimensional aggregate queries efficiently, for example the data cube operator [13]. OLAP systems perform queries fast by pre-computing all or part of the data cube [15]. Such pre-computation takes a large amount of space and is possible only when the query hierarchy along each dimension of interest is fixed or discretized beforehand. For example, a retailer may group stores by city, then region, then state or province and finally by country. All future queries issued by the retailer can group by only the specified categories, if the benefits of pre-computation are to be realized.

While this works well when the query categories are fixed beforehand, there are many cases where it is too restrictive. Consider an employee database in which each employee has attributes – age, salary and vested amount in a pension plan. A user may want to issue a query that provides a cross-tabulation of the average ages of employees having pension plans in the range 100K-200K, 200K-400K, 400K-1000K and having salaries in the range 50K-100K, 100K-200K, and 200K-300K. For traditional approaches, the problem is that the ranges specified by the user can be arbitrary. In other words, the query hierarchy is dynamic and not pre-discretized along each dimension.

Existing approaches are also limited in that the aggregation dimension must be fixed beforehand. Thus, if a cube is materialized for age as the aggregate dimension, then it can only be used for aggregate queries on age. It cannot be used to query the average salary of people in a particular age group. Efficient processing of such queries requires the materialization of more (partial) data cubes, resulting in higher space overhead. Since cube size explodes as dimensionality increases, even a single (partially) pre-computed data cube has significant storage requirements [23]. Hence cubes must be limited to only a few dimensions.

1.1 Contributions of this Paper

We propose a new cube compression technique that is based on modeling the statistical structure of the data. By estimating the probability density of the data, we can build an extremely compact data representation that supports aggregate queries. This reduction has the following desirable properties:

1. The ability to efficiently answer *ad hoc* queries over continuous dimensions without pre-discretizing each such dimension.
2. The ability to store a compact, portable representation of the database that is smaller than the original database by orders of magnitude, and much smaller than the corresponding cube. This enables ad hoc query support on small or mobile clients (for example, laptops) without the need to be connected to the main OLAP server.
3. The ability to build cubes on many dimensions without incurring a large storage penalty.
4. The ability to treat aggregate dimensions interchangeably while using several aggregate measures without paying an added storage penalty.

The tradeoff is *accuracy*: approximate, rather than exact results. This is appropriate for most exploratory data analysis applications and cube design exercises. For example, an administrator may wish

to explore many possible cubes before committing to the expensive task of materializing an actual cube. Also, in reporting applications, it may be desirable to provide off-line support with approximate results while the report is being created and edited. Actual values can be populated prior to publishing the final report by synchronizing with the database server and performing the expensive queries. A bound on the error of this scheme is straightforward and we present a technique to approximately minimize it.

This work exploits the probability density distribution of the data and our approach to estimating this density is based on statistically clustering the records in the database. Clustering provides a mixture model density estimate, which is compact yet capable of supporting aggregate queries (count, sum, average, etc.) We show that it is possible to support such queries while utilizing a very small memory requirement. The primary emphasis is on studying the behavior of our scheme at extreme levels of compression. We then describe how more memory can be utilized.

We consider continuous-valued dimensions because the issues of dynamic query hierarchy and interchangeability of the aggregation variable are most relevant in this situation. We believe, however, that the same approximation technique can be used for data having discrete-valued dimensions (see Section 6). This paper introduces the notion that statistical models of the data can be used to dramatically reduce the size of data cubes while supporting approximate aggregate queries. It is not our intent to replace cubes, nor do we claim that the method used to estimate the probability density of the data is the best one possible: we aim at a proof of concept.

1.2 Related Work

As mentioned above, much work has been done on providing exact answers to queries over the data cube [15][27]. All such approaches share the disadvantages of excessive space overhead, pre-discretizing dimensions and treating dimension and target measures non-uniformly. More recently, work has been done on approximating data cubes through various forms of compression such as wavelets [25] and multivariate polynomials [3]. A related approach is that of multi-dimensional histograms [20]. Beyer et. al. [4] consider the problem of pre-computing a sparse data cube for queries accessing more than a certain number of data records using association rule mining techniques. All these approaches reduce the space overhead but still have the stringent restrictions of pre-discretizing dimensions and are tailored to a particular target dimension and aggregate function. To the best of our knowledge, there is no general technique proposed that solves the range of practical problems addressed in this paper.

Work on efficient computational schemes for exact query support includes [9][18][27][14][15][19][1]. In [9], a scheme is proposed in which portions of previous queries are cached and reused. The work of [27] is focused on optimizing multiple related dimensional queries. In [18], an alternative storage and index organization scheme is proposed based upon a collection of packed and compressed R-trees. In [15], the focus is on determining which cells to pre-compute in a cube in order to reduce size, yet maintain ability to infer the missing cube values using logical or algebraic constraints. This approach is restricted to the traditional OLAP framework where all dimensions are discrete.

In contrast, this work targets a dramatic reduction in storage while supporting approximate query results over continuous-valued data attributes. As such, it addresses a major gap in actual OLAP systems today. It is also a direct application of scalable clustering techniques in Data Mining.

2. ANSWERING AGGREGATE QUERIES USING DENSITY DISTRIBUTIONS

Multi-dimensional data records can be viewed as points in a multi-dimensional space. For example, the records of the schema (age, salary) could be viewed as points in a two-dimensional space, with the dimensions of age and salary. Figure 1 shows some data conforming to the above example schema. Figure 2 shows its representation as points in a two dimensional space.

AGE	SALARY
25	50000
28	55000
30	58000
50	100000
55	130000
57	120000

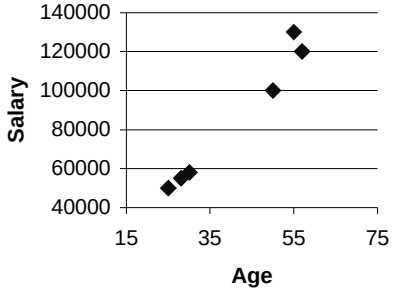

Figure 1: Sample Data **Figure 2: 2-D View of Data**

Aggregate queries over multi-dimensional data sets specify a region in the multi-dimensional space and an aggregation function of interest. For example, an aggregate query may require the number of data records in the region $10 \leq$ age ≤ 40, and $30K \leq$ salary $\leq$ 80K. The aggregate function in the above example is "count" – the number of data records in a particular region. Common aggregate functions include *sum* (the sum of values, in a particular dimension, of data records in a region) and *average* (the ratio of the sum and count aggregate functions).

A key observation is that if the data probability density distribution is known, multi-dimensional aggregate queries can be answered without accessing the data. As a concrete example, assume that the data density function for the two dimensional case with points having age a and salary s is given by $\Pr(a,s)$. Then, assuming that $\Pr(a,s)$ is integrable, one can compute the aggregate query count, for $a_1 \leq a \leq a_2$ and $s_1 \leq s \leq s_2$ as:

$$N \cdot \int_{s_1}^{s_2} \int_{a_1}^{a_2} \Pr(a,s) \cdot da \cdot ds$$

where N is the total number of records in the data set. Similarly, the sum of the ages for data with $a_1 \leq a \leq a_2$ and $s_1 \leq s \leq s_2$ can be computed as (computing the sum of salary is symmetric):

$$N \cdot \int_{s_1}^{s_2} \int_{a_1}^{a_2} a \cdot Pr(a,s) \cdot da \cdot ds.$$

The average age for data with $a_1 \leq a \leq a_2$ and $s_1 \leq s \leq s_2$ can be computed as the ratio of the above two quantities (computing the average of salary is symmetric). There are several advantages to

executing the query using the density function Pr(*a*,*s*) rather than the actual data:

- If the density function is compact, significant storage is saved as the actual data is not used to answer the queries.
- If the integration of the density function is efficient, then executing queries in this fashion is efficient and presents an alternative to pre-computation without the disadvantage of pre-discretizing query hierarchies.
- The same density function is used to answer many different aggregate queries (i.e. any dimension can also be a *measure* in OLAP terminology), without paying an added cost, leading to a further savings in space over traditional pre-computation approaches.

Thus a density-based approach addresses the limitations of existing pre-computation techniques.

There are two crucial properties that the density function must satisfy in order to realize the benefits: (a) the density function must be compact; and (b) integration of the density function must be efficient. The challenge is to derive such a density function from large volumes of data.

Techniques for estimating densities include: histogram-based methods, Parzen windows, and kernel-density estimates [21][24]. Kernel density estimators place a kernel (typically a bump-shaped function; e.g. a Gaussian) atop each data point. Assuming that the kernel has the appropriate width, the sum of the kernel contributions at any point can be shown to converge to the value of the true density function as the number of data points tends to infinity [24]. Kernel-based methods are essentially nearest-neighbor-type algorithms: assuming far-off points have negligible contribution to the sum, one only has to find the nearest neighbors and sum over their kernel contributions to obtain the density estimate at the point. Kernel-based density estimates are extremely difficult to compute in high dimensions [21][24]. In addition, they require the presence of the data set, and hence are not effective for compression. We employ clustering as our basic density estimation method.

3. DENSITY ESTIMATION USING CLUSTERING

In this paper we choose to use clustering-derived mixture models as our efficient and scalable approach to density estimation. The clustering problem has been formulated in various ways in statistics [2][16][24][21], pattern recognition [5][10][12], optimization [6][22], and machine learning literature [11]. The fundamental problem is that of grouping together (clustering) data items that are similar to each other. Data is generally not uniformly distributed and some combinations of attribute values are more likely than others. Clustering can be viewed as identifying the dense regions of the probability density of the data source. An efficient representation of the probability density function is the *mixture model*: a model consisting of several components (e.g. a model consisting of the sum of 3 Gaussians). Each component generates a set of data records (a "cluster"). The data set is then a mixture of clusters and the problem is to identify the data points constituting a cluster and to infer the properties of the distribution governing each cluster.

The *mixture model* probability density function has the form:

$$\Pr(\mathbf{x}) = \sum_{\ell=1}^{k} W_\ell \ \Pr(\mathbf{x}|\ell).$$

The coefficients W_ℓ (mixture weights) are the fraction of the database represented by the corresponding clusters, and k is the number of clusters. We focus on models whose components (clusters) are represented by multivariate Gaussians, though in principle any representation that can be integrated numerically can be used (see Section 3.2). This choice of Gaussians is motivated by the fact that *any distribution can be approximated accurately with a mixture model containing a sufficient number of components* [24][21]**.** Further, clustering is ideal for the following reasons:

- Clusters represented by multivariate Gaussians are compact to represent and easily integrable.
- Efficient methods for clustering large volumes of data have been developed [7][8][26].

Next we discuss how queries can be efficiently evaluated. The following discussion does not depend on a particular clustering method. The only requirement is that the clusters be represented in a compact form that can be efficiently integrated.

3.1 Representation of Clusters

Each cluster in a mixture model is represented by a multivariate Gaussian probability density function:

$$\Pr(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\sqrt{|\Sigma|}} e^{\left(\left(\frac{-1}{2}\right)(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}$$

where $\mathbf{x} = (x_1,x_2,x_3,x_4,\ldots,x_n)$ is a n-dimensional column vector corresponding to a data point in the selected n-dimensional data space, $\boldsymbol{\mu}$ is the n-dimensional column vector whose elements are the means (averages) of the data belonging to the cluster. Σ is an $n \times n$ covariance matrix, with determinant $|\Sigma|$ and matrix inverse Σ^{-1}.

For n-dimensional data, a cluster is represented by $1 + n + [n(n+1)]/2$ values: the number of data points in the cluster (1), the mean vector $\boldsymbol{\mu}$ (n), and symmetric covariance matrix Σ ($n(n+1)/2$). If the covariance matrix is diagonal, then its representation requires only n values, and the resulting cluster is characterized by $2n+1$ values. For a mixture model with k clusters, the total representation cost is $k(1+2n)$. We assume the diagonal covariance structure in this work.

3.2 Answering Aggregate Queries using Cluster Representations

We consider three types of aggregate queries: count, sum and average. We first describe how range selection queries are answered, before outlining how multiple intervals can be specified along a each dimension.

Count Queries: Suppose a query requests the number of items (count) in specified ranges on variables x_{i1} through x_{im}, ($m < n$, i.e. a subset of the n dimensions), the ranges being from a_r to b_r for dimension x_{ir}. Let the unspecified dimensions be x_{j1} through x_{jn-m}.

In this case, count = $\sum_{l=1}^{K} Num(l)$, where

$$Num(l) = N_l \cdot \int_{-\infty}^{\infty}\ldots\int_{-\infty}^{\infty}\int_{a1}^{b1}\ldots\int_{am}^{bm} \Pr(x_1, x_2, \ldots, x_n \mid l)\, dx_{im}\ldots dx_{i1} dx_{j1}\ldots dx_{jn-m}.$$

Assuming a diagonal covariance matrix, Pr(**x**) can be expressed as a product of univariate Gaussians. Hence, the relation for *Num(l)* becomes:

$$Num(l) =$$

$$N_l \cdot \int_{-\infty}^{\infty} \Pr(x_{j1} \mid l)dx_{j1} \cdot \int_{-\infty}^{\infty} \Pr(x_{j2} \mid l)dx_{j2} \ldots \int_{-\infty}^{\infty} \Pr(x_{jn-m} \mid l)dx_{jn-m} \cdot \int_{a1}^{b1} \Pr(x_{im} \mid l)dx_{im} \ldots \cdot \int_{a1}^{b1} \Pr(x_{i1} \mid l)dx_{i1}.$$

The integrals from $-\infty$ to $+\infty$ for dimensions not involved in the range queries evaluate to one. The remaining terms are univariate integrals over the dimensions x_{i1} to x_{im}.

Sum Queries. This query specifies ranges in dimensions x_{i1} through x_{im} and requests for the sum of data items present in the range over a specific dimension x_S. The range specified on variable x_{ir} is $[a_r, b_r]$. Using the above notation, let the unspecified dimensions be represented as x_{j1} through x_{jn-m}. Again, Sum = $\sum_{l=1}^{K} Sum(l)$, where for each l,

$$Sum(l) =$$

$$N_l \cdot \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \int_{a1}^{b1} \ldots \int_{am}^{bm} x_S \Pr(x_1, x_2, \ldots, x_n \mid l)dx_{im} \ldots dx_{i1} dx_{j1} \ldots dx_{jn-m}.$$

The integral decomposes and can be evaluated as before.

Average Queries. This query can be computed as the ratio of the result of the sum query for dimension x_s in the specified ranges and the result of the count query in the specified ranges.

Multiple Interval Queries. In the discussion above, we assumed that only one range selection is specified along each dimension. Disjunctive queries are easily transformed to sums over disjoint ranges. Another common type of query is a "cross-tabulation", where multiple ranges are specified along each dimension. Thus, a query may require the number of data points for every combination of ages in the ranges 10-20, 20-30 and 30-40 and salaries in the range 50K-60K, 60K-90K and 90K-120K. Rather than evaluating all combinations (in this case, there are nine) as separate queries, integrals corresponding to sub-queries may be cached and re-used. Thus, in the current example, the integral for the age ranges 10-20, 20-30 and 30-40 would be performed exactly once for the entire query.

4. THE BASIC ALGORITHM

The first algorithm we investigate takes as input the memory budget and attempts to fit the compressed data representation into it. Since we are interested in studying extremes of compression, we focus on building small models of large databases as proof of concept.

Consider the representation of data as a set of clusters. At one extreme, one can assign an individual cluster to each data point. In this case there is no compression and answers to queries are exact. This approach is tantamount to holding the entire database in memory and clearly does not scale. As the memory bound is reduced, one needs to select "good clusters" to accurately represent substantial subsets of the data. Assuming each cluster represents the data it displaces accurately, the accuracy lost due to dropping data records is (hopefully) minimized.

Recall that the storage requirement for the mixture model with k diagonal Gaussians is $k(2n+1)$ values as opposed to Nn values to represent N data records. We assume that clustering is performed independently (e.g. scalable EM clustering [8]) and that we are given a mixture model consisting of a set of k' clusters. If the memory budget only allows $k < k'$ clusters, the k clusters with maximum data representation are selected. If $k > k'$ either: the dataset is re-clustered with k clusters, or if re-clustering is not a viable option, the remaining memory buffer is filled with data points fitting the current model least (i.e. the buffer is filled with outliers). An outlier is a data point that gets assigned the lowest maximum density by the k clusters.

When answering a query, integration is done over the model, and then the outliers are checked to see if any contribute to the answer (i.e. the aggregation is performed over models plus data). Answers obtained over actual cached records are by definition exact.

We are concerned with evaluating the concept of supporting count, sum and average queries over a compressed data model. We focus on the case when $k' > k$. Performance is evaluated in Sections 4.3 – 4.5. In Section 5 we introduce an algorithm that grows additional clusters in the case when $k' < k$.

4.1 Real World Data Sets for Empirical Evaluation

The four data sets in Table 1 are used in evaluating performance. Since OLAP type queries are usually performed over low-dimensional data sets, we select 3 and 5 dimensional projections. This allows us to study the effect of varying k, and concentrate on using small memory buffers. High-dimensional datasets typically require many more clusters to be modeled accurately[1] [5].

For each data set we construct a statistical model using the EM algorithm [8]. Model construction details are not relevant. Cluster models used in the evaluations in this section were computed for fixed values of k and were initialized randomly on the range of the data. These models were not tuned to specifically support the query application.

The performance measures were:

- size of the model used compared to data size (compression ratio),
- accuracy of the model as compared to performing the queries over the actual SQL database,
- and model query response time as compared to time taken by a DBMS.

For the DBMS, all experiments were conducted using Microsoft SQL Server version 7.0 running on a Pentium II 300 MHz workstation with Microsoft Windows NT 4.0 Server OS.

[1] Fortunately, experiments indicate that scalable EM degrades more gracefully as dimensionality increases than other variants of EM targeted at massive databases [8].

Data Set	Description	Size	Dim	Projections
Census3, Census5	U.S. Census Bureau "Adult" database. Publicly available from SGI [17] (obtained via public ftp from www.sgi.com/Technology/mlc/db/).	299281	11	3 and 5
Astro3, Astro5	Astronomy data representing features of sky objects measured in an image. Provided by Caltech.	648291	29	3 and 5
Investor	U.S. stock data (market cap, earnings, etc.)	835600	34	3
Cover3, Cover5	Forest covertype data (publicly available at www.ics.uci.edu/~mlearn/MLSummary.html)	581012	10	3 and 5

Table 1: Experimental Real Data Sets

Dataset	Ave. SQL Time (s)	Ave. Cluster Time (s)	**Time Ratio**	**Compression Ratio**
Census3	1.4	0.0009	**1556**	**1283**
Census5	1.7	0.001	**1700**	**1360**
Astro3	17.5	0.0005	**35000**	**2778**
Astro5	18.7	0.0009	**20778**	**2950**
Investor	4.4	0.0003	**14667**	**3584**
Cover3	8.4	0.0008	**10500**	**2488**
Cover5	8.1	0.001	**8100**	**2639**

Table 2: Evaluation time and compression ratios.

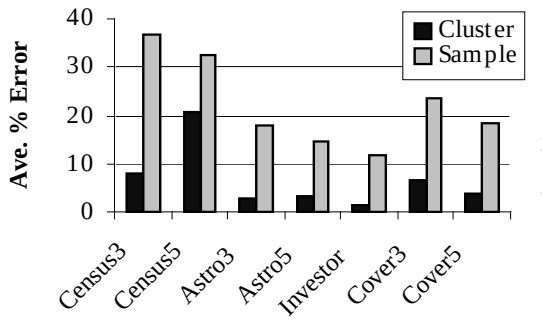

Figure 3: Average % Error for Cluster and Sample.

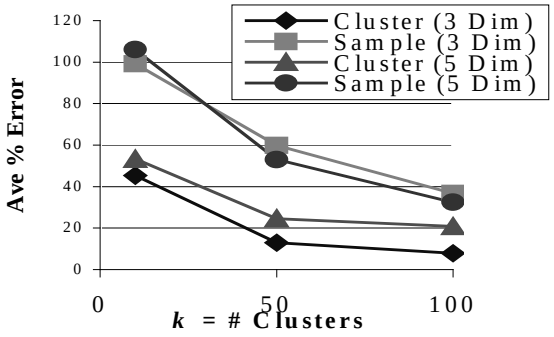

Figure 4: Varying *k* on Census dataset.

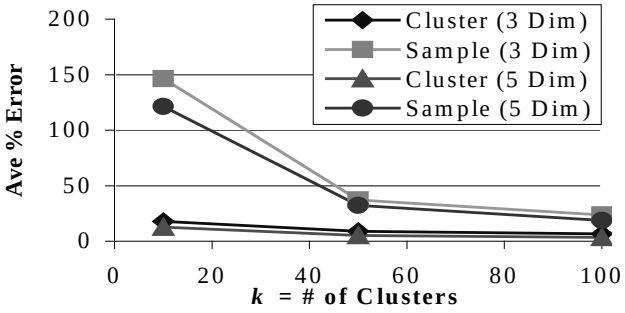

Figure 5: Varying *k* on Cover Dataset

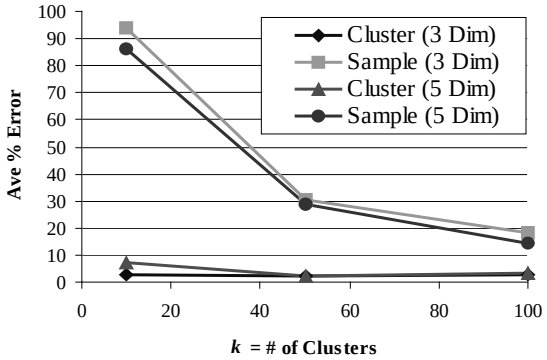

Figure 6: Varying *k* on Astro dataset.

4.2 Experimental Methodology

Five hundred uniform random queries were generated for each dataset. The queries were equally divided among those accessing 1, 2 and 3 dataset dimensions simultaneously. Results over the cluster model (Cluster) are compared with SQL results over the full database and SQL results over a random data sample (Sample) equal in size to the cluster model. For a given number of clusters *k*, $\frac{k(2n+1)}{n}$ records are randomly sampled from the dataset and the queries are issued over this sample. The result is then multiplied by $\frac{N}{\frac{k(2n+1)}{n}} = \frac{Nn}{k(2n+1)}$, scaling the result to the full database of *N* records.

4.3 Evaluation Time/Compression Ratios

Table 2 summarizes the compression ratios for cluster models with $k = 100$ clusters and evaluation times for queries over the full SQL database and over the cluster model. For datasets in 3 dimensions, 5.6 kB of memory was required to store the mixture model summarizing the dataset. For datasets in 5 dimensions, 8.8 kB was required to store the mixture model.

4.4 Measuring Accuracy

We report accuracy results for the datasets listed above using cluster models with $k = 100$. Average relative errors of Cluster and Sample query results with respect to the SQL result over the full dataset are summarized in Figure 3. An improvement of as much as 87% is observed on the Investor dataset.

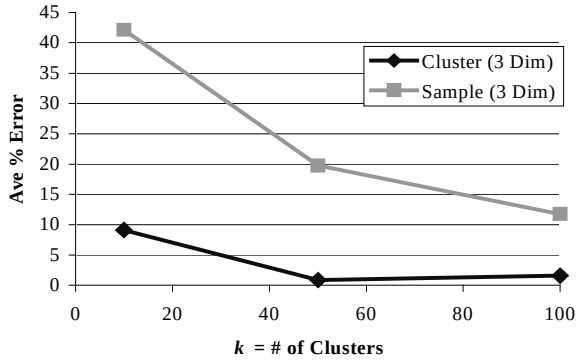

Figure 7: Varying *k* on Investor dataset.

4.5 Effect of Increasing *k*

Figures 4-7 summarize the effect of varying the number of clusters *k* on the query error. Average percent error with respect to the SQL result on the full database is plotted for cluster models with the number of clusters $k = 10, 50, 100$. For comparison, results over random samples of equivalent size are included (Sample). These results indicate that indeed, on average, as the cluster model complexity is increased (via increasing *k*), the resulting mixture model more accurately approximates the data density[2]. The cluster models for all values of *k* tested yielded superior query results in comparison to the sampling approach. The following table summarizes the memory requirement (in kilobytes (kB)) needed to store the mixture models for each dataset used in this evaluation.

Dataset	Model Memory Requirement		
	$k = 10$	$k = 50$	$k = 100$
Census3 Astro3 Investor Cover3	0.56 kB	2.8 kB	5.6 kB
Census5 Astro5 Cover5	0.88 kB	4.4 kB	8.8 kB

4.6 Discussion

The results presented were over real-world data sets. We have experimented with synthetic data sets, and obtained very good results as expected. We note that in general, we do expect our method to perform poorly when the actual query result is small. For small query results, the distinction between zero and a small number is not meaningful. In a practical application, large error bars would be associated with estimates of small counts.

It is difficult to measure an error for queries in which SQL returns 0.0. In this case, we measured the percentage of queries that Cluster returns 0.1 or less. These ranged from 24% (Census3) to

[2] Small variations to this trend are sometimes present, like in Figure 7 ($k = 50$ and $k = 100$), because of the random choice of initial seed values that affect the quality of the clusters.

75% (Cover3). Results presented in Sections 4.3 and 4.4 are averaged over queries in which the full SQL result is greater than 1% of total data set. The following table summarizes average size of the SQL result for the databases used in the evaluation.

Dataset	Ave. SQL Result Size
Census3	12739
Census5	14034
Astro3	51155
Astro5	58265
Investor	71599
Cover3	30982
Cover5	37337

It is important to note, however, that the cluster models used in the evaluations in this section were computed for fixed values of *k*, and were initialized randomly on the range of the data. These cluster models were not tuned to specifically support our query application. This issue is addressed in the next section.

5. MODEL GENERATION TO SUPPORT QUERYING

In the previous section, we outlined an approach to approximately answer multi-dimensional aggregate queries given a probabilistic model of the data. The accuracy of the answer depends critically on the fit of this model to the dataset. We define a notion of accuracy appropriate in our context, and then outline a technique that approximately satisfies this definition and also handles incremental updates.

5.1 A Definition of Accuracy

The notion of accuracy that we consider is defined for the result of an aggregate query. There are several components to the accuracy metric. We describe and motivate each in turn.

a) **Deviation from the Actual Value (*d*):** This requirement is the percentage difference between the approximate query result and the actual query result.

b) **Confidence (*c*):** This requirement specifies the percentage of time that the approximate query result satisfies the deviation requirement.

c) **Support for the Aggregate Result (*s*):** This accuracy requirement is used to model the nature of decision support queries. Most queries deal with aggregates over large amounts of data, not with selections of individual data points. This parameter establishes a threshold on the number of data points such that only queries that select more than *s* data points need to satisfy requirements a) and b).

d) **Number of Outliers (*o*):** One important goal of decision support queries lies in detecting information that is unusual. This parameter specifies the number of "unusual" data points that are to be identified and explicitly stored.

Suppose an approximate model was defined with the following values for the parameters: d = 10%, c = 90%, s = 1000, o = 50. This model then answers all queries that aggregate over more than 1000 data points within 10% accuracy 90% of the time. In addition, queries whose selection conditions specify aggregates over any subset of the 50 most unusual points will return exact results.

Note that conditions a), b) and c) progressively weaken the accuracy requirement. If d were set to 0%, c to 100% and s to 0, then the accuracy metric would not allow for any approximation. The factor d) explicitly models a common decision support requirement. Increasing factor d) increases the accuracy of the model because more data points are explicitly stored.

In the next section, we outline a technique to build a cluster model that approximately satisfies these requirements.

5.2 Constructing Cluster Models Satisfying Accuracy Requirements

The goal in this section is not to propose a new clustering technique. Rather, our focus is to exploit existing clustering strategies and to adapt them for our purposes. We base our discussion on the Expectation-Maximization (EM) clustering algorithm [8].

The idea is to iteratively refine the cluster models until the accuracy requirements are satisfied. This is done by identifying data regions in which the current cluster model violates the accuracy requirements. New clusters are grown in these regions. The high-level algorithm for the technique is as follows.

```
C := Initial Cluster Model

While (C is not sufficiently accurate) do
    Grow new clusters in C where it is
    not sufficiently accurate
End while

C is the required cluster model
```

The initial model can be chosen based on some standard clustering mechanism [7][8][26]. The challenge lies in determining when a cluster model is not sufficiently accurate and then to grow new clusters so that the new model is more accurate. We address these two issues next.

5.2.1 Determining and Improving the Accuracy of a Cluster Model

The problem is identifying whether the cluster model satisfies the accuracy parameters. This is achieved by dividing the multidimensional space into (possibly overlapping) tiles. The number of data points belonging to a tile is set approximately equal to the support for the aggregate result (the accuracy parameter *s*). The motivation behind this approach is that the tiles serve as surrogates for possible range queries over the data. Data points in the tiles correspond to those selected by an aggregate range query. By evaluating the accuracy criterion on the individual tiles, an estimate of the accuracy of the model on queries selecting more than *s* data points can be calculated. We next address the issue of placement of the tiles.

One of the key roles of the tiles, in addition to determining the accuracy of the model, is to identify the location where new clusters are to be grown. If a cluster does not sufficiently summarize the data belonging to it (according to the specified values of *d*, *c* and *s* (Section 5.1)), the cluster needs to be split. Tiles are placed according to the current clustering model. Each

cluster is divided so that the area under the Gaussian is the same for each tile. Intuitively, each tile then should encompass the same number of data points, *if the data follows the Gaussian distribution of the cluster*.

As an illustrative example, consider a one-dimensional Gaussian cluster which summarizes 3*s* data points. The tiling procedure would result in three tiles, each having area under the Gaussian of 1/3 (i.e. *s* points). One such tile will have range from $-\infty$ to *t* such that the area under the Gaussian on this range is 1/3. Similarly, the two other tiles have ranges [*t*,*u*] (*t* < *u*) and [u, $+\infty$) such that the area under the Gaussian on these ranges is 1/3, respectively. The above three ranges correspond to tiles for the one-dimensional Gaussian given the correctness requirements. This is done in multiple dimensions (assuming low dimensionality) independently – justified by the diagonal covariance structure.

We now determine whether each tile satisfies the accuracy requirement. The data records are scanned and partitioned first by cluster membership, then by tile membership within the cluster. Let *N* be the total number of data points in the dataset and let *N(l)* be the number of data points with membership in cluster *l*. Let $N_T(l)$ be the number of data points contained within tile *T* of cluster *l* defining a region in the *n*-dimensional data space. Let Pr(**x** | *l*) denote the Gaussian density function for cluster *l*. The error for tile *T* is then defined as the difference between the model-predicted data density in the tile and the actual data density in the tile: $E(T) = N_T(l) - N\int_T \Pr(\mathbf{x} \mid l)d\mathbf{x}.$. Tile *T* satisfies the accuracy requirement if $\left|\frac{E(T)}{N}\right| < \frac{d}{100}$.

Let $\Gamma(l) = \{ T_1, T_2, \ldots T_h \}$ be the set of tiles for cluster *l*. Let $\Gamma_S(l)$ be the subset of $\Gamma(l)$ satisfying the accuracy requirement: $\Gamma_S(l) = \left\{ T \in \Gamma(l) \;\middle|\; \left|\frac{E(T)}{N}\right| < \frac{d}{100} \right\}$.

A cluster *l* is said to satisfy the accuracy requirement if: $\frac{|\Gamma_S(l)|}{|\Gamma(l)|} < \frac{c}{100}$, where $|\cdot|$ applied to a set indicates the number of elements in the set. Hence cluster *l* satisfies the accuracy requirement if *c*% of its tiles satisfy the requirement. The clusters that do not satisfy the accuracy requirement are split into smaller clusters, each of which will hopefully better satisfy the accuracy requirements.

For each cluster that does not satisfy the accuracy requirement, the tile having the maximum positive error (or the tile having the minimum negative error if a tile with positive error does not exist) is determined. A positive error over tile *T* occurs when *T* contains more data than the model predicts (i.e. $N_T(l) > N\int_T \Pr(\mathbf{x} \mid l)d\mathbf{x}$). Similarly, a negative error occurs when the tile contains less data than model predicts (i.e. $N_T(l) < N\int_T \Pr(\mathbf{x} \mid l)d\mathbf{x}$). The model approximates the actual data distribution worst on this tile. We place a new cluster at this location. Such new cluster locations are determined for each offending cluster and the clustering algorithm is run again, with new cluster initialization points in addition to the original clusters. The RefineModelAccuracy (RMA) algorithm implementing this process is summarized below and is executed repeatedly until the accuracy requirement is satisfied. The specified number of outliers *o* are then found in a single data scan.

5.2.2 Incremental Updates

Updates can be handled efficiently within the RMA framework. The tiling procedure is used to determine whether the updates violate the accuracy requirement of the cluster model. If the accuracy requirement is satisfied, then no change is required to the cluster representation. On the other hand, if the accuracy requirement is not met, clusters are grown in only those regions where the discrepancy exists as in the regular RMA algorithm.

Algorithm RefineModelAccuracy (RMA)

(Input: Model *C*, Support *s*, Deviation *d*, Confidence *c*, Data *D*) Output Model *C'*

1. Split each cluster in the model *C* into tiles each of size approximately *s*
2. Scan the data *D* and classify each data point first by the cluster membership and then by the tile membership in the cluster it belongs to. Also, update the aggregate value for each tile
3. Set Tile Set *T* = $\varnothing$
4. For each cluster in *C* do
   If the aggregate value for *c*% of the tiles in the cluster is within *d*% of the expected value *s*,
   THEN the cluster is accurate.
   ELSE add the tile having maximum positive (minimum negative) error to the tile set *T*
5. If the tile set *T* = $\varnothing$,
   THEN cluster model *C* satisfies the accuracy requirement. Exit.
   ELSE
   a. Rerun the clustering algorithm with the mid-points of the tiles in *T* and the means of clusters in old model *C* as the new cluster initialization.
   b. Repeat steps 1 – 5 for new model.

5.3 Experimental Evaluation

Our goal in this evaluation is not to exhaustively test the algorithm, but to demonstrate that refinement is possible and results in improved models of the data. Cluster models were computed via algorithm RMA with the following fixed parameter settings: deviation d = 10%, confidence c = 90%, support s = N/3125 for datasets with N records and 5 dimensions and s = N/1000 for datasets with 3 dimensions, in all cases the number of outliers o = N/1000. These conservative values for s were chosen so that tiling the 5 dimensional datasets results in approximately 5 partitions per dimension and 10 partitions per dimension for the 3 dimensional datasets. The experimental setup was the same as in Section 4.2.

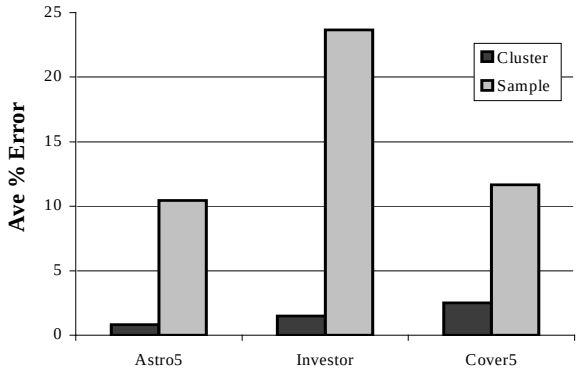

Figure 8: Cluster models computed by RMA.

Average error percentages with respect to the true SQL result for cluster models (Cluster) and random samples (Sample) are given in Figure 8 for Astro5, Investor and Cover5 datasets. The results are averaged over queries in which the full SQL result is greater than 1% of total data set. Resulting cluster models had the following number of clusters: k = 219 for Astro5, k = 91 for Investor and k = 318 for Cover5. The memory requirement to store the mixture model for Astro5 was 19.3 kB and requirements for Investor and Cover5 were 5.1 kB and 28.0 kB, respectively. These cluster models provide superior results over sampling and improved accuracy by as much as 93% (Investor) over the sampling result with a corresponding compression ratio of 3935. In comparison to cluster models used in Section 4.4, models produced by RMA improved query results by as much as 76% (Astro5). RMA models improved Investor results by 6% and Cover5 results by 30% over those used in Section 4.4. Average SQL result for the Astro5, Investor and Cover5 datasets were 58265, 71559 and 37337, respectively.

A more exhaustive evaluation of RMA studying its sensitivity to parameter settings and performance over other large databases is not presented here due to space limitations. However, the goal of this paper is to demonstrate that the concept is feasible, and that reasonable accuracies can be obtained with 3 orders of magnitude reduction in space over the size of the data. We have purposefully limited our exposition to studying the performance of small model.

6. CONCLUSIONS AND FUTURE WORK

This work demonstrates that significant compression ratios are attainable while supporting OLAP-type aggregate queries over large databases when the statistical structure of the data is exploited. In addition, the method presented supports *ad hoc* querying over continuous dimensions, allows each dimension to serve as a measure and supports various aggregate functions. In general, OLAP systems do not support continuous dimensions without pre-discretization - hence our scheme fills an important niche in the OLAP space. Also notable is the fact that our approach performs significantly better than a sampling-based approach - this is not surprising but serves as additional evidence supporting the merit of the technique introduced here.

The tradeoff when using this compression scheme is a sacrifice in accuracy. Empirically, however, accuracy is not exceedingly compromised. Moreover, the Gaussian mixture model readily gives rise to error bounds on the approximate query result. A system utilizing this technique is capable of judging uncertainty associated with a prediction and this information can be used to enhance usability. In particular, inaccuracy occurs when the query result is small and the system can suppress or highlight such results to the user.

We introduced an algorithm dedicated to the task of computing cluster models specifically supporting OLAP-type queries, rather than using general models obtained by classical EM clustering. Initial computational evaluations of this algorithm were provided.

We plan to follow many threads as future work. An obvious extension is to generalize to discrete dimensions using discrete clustering techniques. Note that the theory makes no assumptions regarding the type of data attributes or the density distribution. The only requirement is that the distribution be efficiently integrable and have a compact representation. For discrete attributes, the multinomial distribution naturally fits in the context of EM clustering and satisfies our distribution requirements.

We also plan to investigate statistical methods that can provide tighter error bounds. For instance, the statistical density model can be used to assess the uncertainty regarding the expected result. The variance can be derived for the predicted quantity over the query region. This allows the system or the user to make tradeoff decisions between accuracy and the utility of having an approximate result quickly. Finally, studying special clustering methods tailored to the task of answering queries is an intriguing problem. The presentation of the RMA algorithm was primarily targeted at demonstrating this possibility.

7. ACKNOWLEDGMENTS

We gratefully acknowledge Jeff Bernhardt and Ilya Vinarsky for much assistance with implementation and other aspects of this work. This work was conducted while J. Shanmugasundaram was on a research internship at Microsoft Research.

8. REFERENCES

[1] S. Agarwal, S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, S. Sarawagi, "On the Computation of Multidimensional Aggregates", *Proc. 22nd Int. VLDB Conf.*, Mumbai (Bombay), 1996.

[2] J. Banfield, A. Raftery, "Model-based Gaussian and non-Gaussian Clustering", *Biometrics*, vol 49:803-821, 1993.

[3] D. Barbara, M. Sullivan, "A Space-Efficient way to support Approximate Multidimensional Databases", George Mason University Technical Report ISSE-TR-98-03, 1998.

[4] K.S. Beyer, R. Ramakrishnan, "Bottom-Up Computation of Sparse and Iceberg CUBEs", *Proc. ACM SIGMOD Conf.*, Philadelphia, 1999 (to appear).

[5] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[6] P. S. Bradley, O. L. Mangasarian, W. N. Street, "Clustering via Concave Minimization", *Advances in Neural Information Processing Systems 9*, MIT Press, 1997.

[7] P. S. Bradley, U. Fayyad, C. Reina, "Scaling Clustering Algorithms to Large Databases", *Proc. 4th Intl. Conf. on Knowledge Discovery and Data Mining (KDD98)*, AAAI Press, 1998.

[8] P. S. Bradley, U. Fayyad, C. Reina, "Scaling EM (Expectation-Maximization) Clustering to Large Databases", Microsoft Research Technical Report MSR-TR-98-35, 1998.

[9] P. M. Deshpande, K. Ramasamy, A. Shukla, J. Naughton, "Caching Multidimensional Queries Using Chunks", *Proc. ACM SIGMOD Conf.*, Seattle, 1998.

[10] R. O. Duda, P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[11] D. Fisher, "Knowledge Acquisition via Incremental Conceptual Clustering", *Machine Learning*, 2:139-172, 1987.

[12] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, San Diego, CA, 1990.

[13] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub Totals", *Data Mining and Knowledge Discovery*, 1(1), pp. 29-53, 1997.

[14] H. Gupta, V. Harinarayan, A. Rajaraman, J. Ullman, "Index Selection for OLAP", *Proc. Intl. Conf. On Data Engineering*, Birmingham, UK, April 1997.

[15] V. Harinarayan, A. Rajaraman, J. Ullman, "Implementing Data Cubes Efficiently", *Proc. ACM SIGMOD Conf*, Montreal, 1996.

[16] L. Kaufman, P. Rousseeuw, *Finding Groups in Data*, Wiley, New York, 1989.

[17] R. Kohavi, 1996. "Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid." *Proc. of the 2nd International Conf. on Knowledge Discovery and Data Mining*. AAAI Press.

[18] Y. Kotidis, N. Roussopoulos, "An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees", *Proc. ACM SIGMOD Conf.*, Seattle, 1998.

[19] I. Mumick, D. Quass, B. Mumick, "Maintenance of Data Cubes and Summary Tables in a Warehouse", *Proc. ACM SIGMOD Conf.*, Tuscon, 1997.

[20] V. Poosala, Y.E. Ioannidis, "Selectivity Estimation Without the Attribute Value Independence Assumption," *Proc. 23rd VLDB Conf.*, Athens, Greece, 1997.

[21] D. W. Scott, *Multivariate Density Estimation*, Wiley, New York,1992.

[22] S. Z. Selim, M. A. Ismail, "K-Means-Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No. 1, 1984.

[23] A. Shukla, P.M. Deshpande, J. Naughton, K. Ramasamy, "Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies," *Proc. 22nd Int. VLDB Conf.*, Mumbai (Bombay), 1996.

[24] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman & Hall, London. 1986.

[25] J.S. Vitter, M. Wang, B. Iyer. "Data Cube Approximation and Histograms via Wavelets," *Proc. 7th Intl. Conf. Information and Knowledge Management (CIKM'98)*, Washington D.C., November 1998.

[26] T. Zhang, R. Ramakrishnan, M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases", *Proc. ACM SIGMOD Conf.*, Montreal, 1996.

[27] Y. Zhao, P.M. Deshpande, J.F. Naughton, "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates", *Proc. ACM SIGMOD Conf.*, Tucson, 1997.

[28] Y. Zhao. P. M. Deshpande, J. F. Naughton, A. Shukla, "Simultaneous Optimization and Evaluation of Multiple Dimensional Queries", *Proc. ACM SIGMOD Conf.*, Seattle, 1998.