Histogram-Based Approximation of Set-Valued Query Answers

Yannis E. Ioannidis[*] [†]
University of Athens
yannis@di.uoa.gr

Viswanath Poosala
Bell Labs - Lucent Technologies
poosala@research.bell-labs.com

Abstract

Answering queries approximately has recently been proposed as a way to reduce query response times in on-line decision support systems, when the precise answer is not necessary or early feedback is helpful. Most of the work in this area uses sampling-based techniques and handles aggregate queries, ignoring queries that return relations as answers. In this paper, we extend the scope of approximate query answering to general queries. We propose a novel and intuitive error measure for quantifying the error in an approximate query answer, which can be a multiset in general. We also study the use of histograms in approximate query answering as an alternative to sampling. In that direction, we develop a histogram algebra and demonstrate how complex SQL queries on a database may be translated into algebraic operations on the corresponding histograms. Finally, we present the results of an initial set of experiments where various types of histograms and sampling are compared with respect to their effectiveness in approximate query answering as captured by the introduced error measure. The results indicate that the *MaxDiff(V,A)* histograms provide quality approximations for both set-valued and aggregate queries, while sampling is competitive mainly for aggregate queries with no join operators.

[*] Partially supported by the members of the Wisconsin database group industrial affiliates program (http://www.cs.wisc.edu/~raghu/dbaffiliates.html) and by the Special Account ($E\iota\delta\iota\kappa o s$ $\Lambda o\gamma\alpha\rho\iota\alpha\sigma\mu o s$) of the Univ. of Athens under grant 70/4/3244.

[†] On leave from the Univ. of Wisconsin - Madison. Part of this work was done while the author was visiting Bell Labs.

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

1 Introduction

The users of a large number of applications pose very complex queries to Database Management Systems (DBMSs), which take a long time to execute. Examples of such applications are decision support, experiment management, etc. Given the exploratory nature of such applications, many of these queries end up producing no result of particular interest to the user. Much wasted time could have been saved if users were able to quickly see an *approximate answer* to their query, and only proceed with the complete execution if the approximate answer indicated something interesting.

It is easy to conceptualize approximating a query answer when that answer is an image; instead of the actual image, a compressed version of it is retrieved. Alternatively, a series of compressed images may be retrieved, each one being less compressed (more accurate) than the previous one, with the last one being the actual image. The key question that we want to answer is how to provide a similar kind of functionality for alphanumeric queries. For ease of presentation, we use the relational model as the underlying environment in this discussion, but the problem and the methodology followed can be applied in more general settings.

Given an SQL query, its answer is a relation, i.e., a (multi)set of tuples (we use the term *set* to mean a *multiset* whenever no confusion arises). Most earlier work in approximate query answering has been dealing with approximating individual values in the results of aggregate queries [1, 6] and does not handle general (non-aggregate) SQL queries. The few instances of past work that have dealt with general queries, have been based on defining approximations as subsets and/or supersets of the actual answer [2]. This is not very useful, however, for many database applications. Much better intuition is given by a set with roughly the same number of tuples as the actual query answer containing values that are approximations of the actual values (e.g., a numeric field having the value 10 instead of 9). This is more apparent when the query result is presented visually, where the analogy to approximating images can be drawn much closer: *a large number of somewhat misplaced points form a more desirable approximation than a small number of the actual points*.

As an example, consider the typical employee relation, and assume that a query asks for the values of the 'salary', 'age', and 'department' attributes. Further assume that the result is to be displayed visually as a set of points (*starfield*) in the salary-age space, each point representing an employee with the corresponding salary and age. The shade of each point represents the corresponding employee's department. A typical display of this information may be as shown in Figure 1.

Consider two approximations to this query answer, as shown in Figures 2 and 3. The first one (Figure 2) is a small subset of the actual answer, e.g., obtained by sample-based query processing aiming at a 20% sample. The second one (Figure 3) is a set with elements close to those of the actual query answer. We believe that, in most cases, users would much rather receive the latter than the former, as it generates a much better feel for the true answer.

The obvious question that arises is how we can obtain query result approximations of the above form. The idea we pursue in this paper is using *histograms* for this purpose. Briefly, we use histograms (in the usual manner) to approximate the data in the database and employ novel techniques to provide approximate answers using SQL queries on the histograms which are stored as relations. A great advantage of taking this route for approximate query answers is that almost all commercial database systems already maintain histograms, so obtaining such approximations does not require any fundamental changes to these systems.

Our contributions are summarized as follows:

- We have defined a novel measure to quantify the error in the approximate answer. Identifying such a measure is essential for any systematic study of approximation of set-valued query answers to take place, because it presents common grounds for comparing multiple techniques. We have shown via a series of examples that this measure provides a satisfactory idea of the quality of an approximation to a general query, while other well known metrics fail to do so.

- We have proposed storing histograms as regular relations in a relational DBMS and appropriately translating regular database queries into equivalent queries on the histograms so that approximate query answers can be obtained using the same mechanism as exact query answers. To this end, we have defined a *histogram algebra* that can be used to express all required queries on histograms, and have implemented its operators in a query processor that we have used for experimentation.

- We have performed an extensive set of experiments comparing various kinds of histograms and sampling with respect to their effectiveness in approximating general query answers. The results point to a specific class of histograms as the most effective overall.

2 Related Work

There are three aspects of this work for which we discuss related work. The first aspect is *approximate query answering*. There has been extensive work on this topic for quite some time now ranging from establishing theoretical foundations [2, 11], to building actual systems (e.g., CASE-DB [12], APPROXIMATE [20]). All these works are based on the subset/superset definition of approximations, which they obtain mostly through partial query processing. Recently, there has been some work on providing approximate answers to *aggregate queries* using precomputed samples [1], histograms [14, 15], and wavelets [19], which does not address general queries. Online aggregation [6] constitutes another style of sampling-based approximate query answering (for aggregate queries) wherein the answers are continuously refined till the exact answer is computed. In contrast to all earlier work, this paper

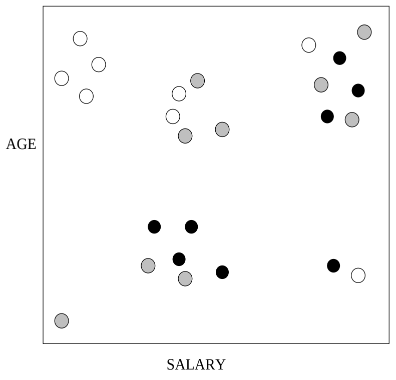

Figure 1: Visual display of query answer

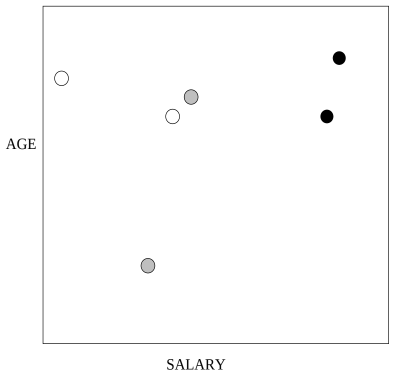

Figure 2: Visual display of approximation of query answer (as a subset)

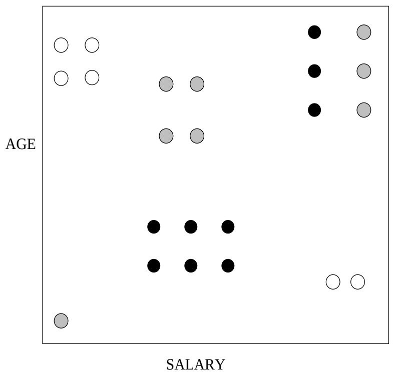

Figure 3: Visual display of approximation of query answer (through value proximity)

deals with producing one approximate query answer to either aggregate or non-aggregate queries.

The second aspect of our work is *statistical techniques*. There has been considerable amount of work in using statistical techniques to approximate data in databases, particularly for selectivity estimation in query optimizers. The three widely studied classes of techniques are *sampling* [10], *parametric techniques* [3] (approximating the data using a mathematical distribution), and *histogram (or non-parametric) techniques* [9, 13, 17]. Of these, histograms are probably the most widely used statistics in commercial database systems (e.g., they are used in the DBMSs of Oracle, Sybase, Microsoft, IBM, etc.), because a) they occupy small amounts of space and do not incur much overhead at estimation time and b) they are particularly suited for accurately approximating the skewed distributions arising in real-life. In our earlier work, we have identified several novel classes of histograms to build on one or more attributes [16, 17] and also proposed techniques for maintaining many of them incrementally up-to-date as the database is updated [5]. However, histograms have not been studied in the context of approximate query answering before.

The third aspect of our work is *quantifying the error in an approximate answer* (a multiset in general). One way to compute this error is to use a known distance metric between two multisets. However, since sets are not embeddable in metric spaces typically, there is very little advanced work in this area. We list three commonly used metrics next. First, it is common to define set-difference based on the cardinality of the symmetric difference between two sets S_1 and S_2, i.e.,

$$dist_{\{\}}(S_1, S_2) \;=\; |(S_1 - S_2) \cup (S_2 - S_1)|.$$

This can be generalized for multisets by making each copy count in the multiset difference. Second, for data distributions, there is the approach based on the various distribution moments. If $F_1 = \{f_{11}, \ldots, f_{1n}\}$ and $F_2 = \{f_{21}, \ldots, f_{2n}\}$ represent the frequency sets of two data distributions on a universe of n elements, the family (for $m \geq 1$) of moment-related distances are as follows:

$$dist_{dd}^{(m)}(F_1, F_2) \;=\; \frac{1}{n}\left(\sum_{k=1}^{n} |f_{1k} - f_{2k}|^m\right)^{1/m}.$$

Finally, the *Hausdorff distance* is another well known metric for comparing two sets [7]. Briefly, two sets are within *Hausdorff distance* h from each other iff any item in one set is within distance h from some item of the other set.

However, none of these approaches capture proximity between the sets in the way we believe is required for approximate query answering. One of the key limitations of the first two metrics is that they do not take into account the actual values of the set elements. For example, according to these two metrics, the sets $\{\,5\,\}$ and $\{\,100\,\}$ are at the same distance from the set $\{\,5.1\,\}$. On the other hand, the Hausdorff metric ignores the frequency of the elements in the sets. For example, the sets $\{\,5\,\}$ and $\{\,5, 5, 5\,\}$ are at the same Hausdorff distance (equal to 0) from the set $\{\,5\,\}$. In contrast, our error measure strikes a balance between values and frequencies, and is therefore more natural for evaluating approximate answers.

3 Error Measure for Set-Valued Approximate Answers

For queries returning a single numerical answer, there is a straightforward metric for the error of an approximate answer. It is simply the difference between the actual and approximate answers. For queries returning a single tuple containing only numerical fields, one can use the *Euclidean Distance* (square root of the sum of squared differences between the corresponding fields). In this section, we develop a novel and robust measure of the error in an answer to a query returning a *multiset* of tuples or numbers. We have examined many formulas before concluding on the one that we present below. The key criterion in our choice has been capturing as much as possible the nature of approximation implied in approximate query answering, where both the actual values in an answer and their frequencies count.

3.1 Error Formula

Consider two multisets $S_1 = \{u_1, \ldots, u_n\}$ and $S_2 = \{v_1, \ldots, v_m\}$ corresponding to the actual and approximate answers, respectively. Let $dist$ be an error metric for the object type of the multiset elements. Based on this, our metric of the error of S_2 with respect to S_1 is computed as follows: first, determine which element in the approximate answer best corresponds to an element in the exact answer and vice versa; then, compute the error by using the $dist$ between the objects mapped to each other.

In more detail, consider a complete bipartite graph G_{S_1,S_2} where the two classes of nodes correspond to the elements of the two answers (each element represented by as many nodes as there are copies of it in the answer). Each edge (u_i, v_j) is associated with a cost equal to $dist(u_i, v_j)$. Let C be a *minimum cost edge cover* of G_{S_1,S_2}, i.e., it is a subset of the edges of G_{S_1,S_2} such that (a) for each node, there is at least one edge in C adjacent to the node, and (b) the following expression is minimized:

$$MINCOVER(S_1, S_2) = \sum_{(u_i,v_j)\in C} dist(u_i, v_j). \tag{1}$$

Then, the error in the approximate answer, which we call its *Match And Compare (or MAC) distance* and denote by $MAC^{(l,m)}$ for any power $l \geq 0, m \geq 0$, is:

$$MAC^{(l,m)}(S_1, S_2) = \sum_{(u_i,v_j)\in C} mult^l(u_i, v_j) \times dist[q]^m(u_i, v_j), \tag{2}$$

where for each edge (u_i, v_j) in C such that d_1 edges are incident to u_i and d_2 edges are incident to v_j, the following hold:

$$mult(u_i, v_j) = max(1, max(d_1, d_2) - 1)$$

$$dist[q](u_i, v_j) = \begin{cases} dist(u_i, v_j), \text{if } max(d_1, d_2) = 1 \\ dist(u_i, v_j) + q, \text{otherwise} \end{cases}$$

Essentially, $mult(u_i, v_j)$ captures the extent to which one of u_i or v_j is paired up with multiple elements of the other multiset[1]. By choosing $l = 0$, one can also opt to ignore the effects of multiplicities, e.g., when interested in the set (not multiset) properties of the two answers. Likewise, q (for $q$uantum, any small integer, typically 1), increases the error due to paired up elements whenever this pairing up is not exclusive on both sides.

Figure 4 shows a small set of examples that illustrate *MAC* and the mapping C. We note that many well-known metrics often failed to capture our intuition behind the error in multiset approximation, whereas *MAC* has worked quite satisfactorily.

Some indication of the naturalness of the formula is that it is symmetric (i.e, $MAC(S_1, S_2) = MAC(S_2, S_1)$) and for certain special cases, it reduces to well-accepted specialized distances and error metrics: for singleton answers, *MAC* is the *dist* between their elements; for frequency distributions with the same set of elements appearing in both (and for $q = 1$), *MAC* is closely related to the corresponding moment-related distance raised to the lth power; and for accurate answers, *MAC* is always equal to zero.

It should be noted that this represents a first attempt at evaluating an approximate answer set. In fact, *MAC* has some drawbacks when used as a general set distance metric. It does not satisfy the *triangle inequality* required of general distance metrics, i.e., $MAC(S_1, S_2) + MAC(S_2, S_3)$ may some times be less than $MAC(S_1, S_3)$. The following counter example shows this: S1 = $\{1\}$, S2 = $\{3\}$, S3 = $\{5, 5\}$. It follows that *MAC*(S1,S3) = 8 + 2q, which is greater than *MAC*(S1,S2) + *MAC*(S2,S3) = 2 + (4 + 2q). Our attempts to fix this deficiency have revealed that *MAC* will have to be significantly simplified in the process, causing it to face the same problems as the other metrics, i.e., not capturing intuition behind approximate query answering. Another minor problem with *MAC* is that in some cases there may be multiple minimum-cost edge covers resulting in different values for *MAC*. In such cases, one should ideally use the least value of *MAC*.

[1]It can be easily shown that $min(d_1, d_2) = 1$ always. If both degrees are greater than 1, one can drop the edge between the two nodes, resulting in a lower cost edge cover.

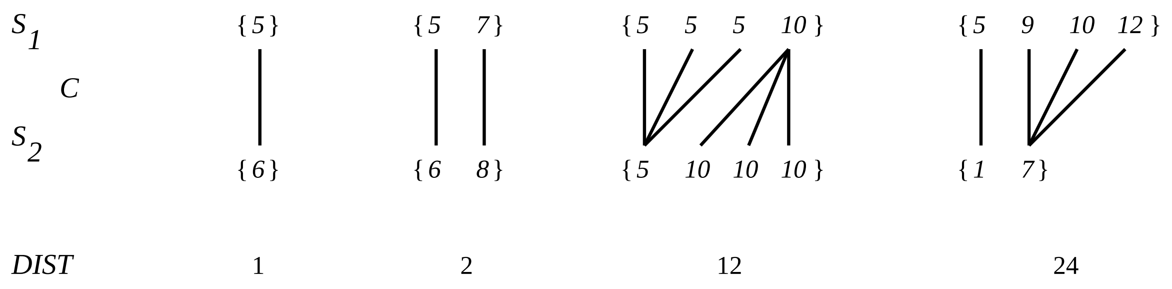

Figure 4: Examples of MAC distances between Answers

3.2 Computation of *MAC*

The difficult part of computing *MAC* is identifying the minimum cost edge cover C between the two answers. For this, we have identified an efficient polynomial time algorithm based on a reduction to the *minimum cost perfect match* problem. The details of this algorithm are given in a related technical report. Here, we present a more efficient heuristic. First, map each element to its closest element in the other set (this is done efficiently by first sorting the two sets). This may result in some surplus edges, i.e., edges that can be removed and still obtain a cover. These are the edges both of whose nodes have degrees greater than 1. Next, go through these edges in decreasing order of their costs (distances) and eliminate each one that remains redundant when it is examined. The complexity of this algorithm is easily seen to be $O(nlogn)$, because of the sorting operations.

4 Histograms

In this section, we give an overview of standard histogram-based techniques for summarizing the data in a database [17, 16]. First, we present some useful definitions.

The *value set* $\mathcal{V}_i$ of attribute X_i of relation R is the set of values of X_i that are present in R. Let $\mathcal{V}_i = \{ v_i(k) \colon 1 \leq k \leq D_i \}$, where $v_i(k) < v_i(j)$ when $k < j$. The *spread* $s_i(k)$ of $v_i(k)$ is defined as $s_i(k) = v_i(k+1) - v_i(k)$, for $1 \leq i \leq D_i$. (We take $s_i(D_i) = 1$.) The *frequency* $f_i(k)$ of $v_i(k)$ is the number of tuples in R with $X_i = v_i(k)$. The *area* $a_i(k)$ of $v_i(k)$ is defined as $a_i(k) = f_i(k) \times s_i(k)$. The *data distribution* of X_i is the set of pairs $\mathcal{T}_i = \{ (v_i(1), f_i(1)), (v_i(2), f_i(2)), \ldots, (v_i(D_i), f_i(D_i)) \}$. Typically, real-life attributes tend to have *skewed* data distributions, i.e., they may have unequal frequencies and/or unequal spreads.

Example 4.1 The following table shows how each parameter defined above is instantiated for a hypothetical attribute.

Quantity	**Data Distribution Element**				
Value	10	60	70	90	100
Frequency	100	120	10	80	2000
Spread	50	10	20	10	1
Area	5000	1200	200	800	2000

A *histogram* on an attribute X is constructed by using a *partitioning rule* to partition its data distribution into β ($\geq$ 1) mutually disjoint subsets called *buckets* and approximating the frequencies and values in each bucket in some common fashion. In particular, the most effective approach for values is the *uniform spread* assumption [17], under which the attribute values in a bucket are assumed to be placed at equal intervals between the lowest and highest values in the bucket. The most effective approach for frequencies is to approximate the frequencies in a bucket by their average (*uniform frequency assumption*). In practice, each bucket in a histogram keeps the following information: the total number of tuples that fall in the bucket (*tot*), and for each dimension i, the low and high values (lo_i, hi_i) and the number of distinct values ($count_i$) in that dimension (the subscripts are dropped for single-dimensional histograms). For the purpose of this work, we store histograms as regular relations in the database with each bucket forming a tuple. For ease of explanation in later sections, we also include additional fields: the average spreads along each dimension ($sp_i = \frac{hi_i - lo_i}{u_i - 1}$) and the average frequency for the bucket ($avg = \frac{tot}{u_1 u_2 \ldots u_d}$).

As an example, consider a 3-bucket histogram on the above data with the following bucketization of attribute values: $\{10\}$, $\{60, 70, 90\}$, $\{100\}$. The buckets in this histogram are given below.

tot	*lo*	*hi*	*count*	*sp*	*avg*
100	10	10	1	-	100
210	60	90	3	15	70
2000	100	100	1	-	2000

Conceptually, one can "expand" a histogram into a relation containing the approximate attribute values as its tuples, with each tuple appearing as many times as the approximate frequency of that value. We call this the *approximate relation* (*ApproxRel*) of that histogram. For a 1-dimensional histogram H, its approximate relation can be computed using the following SQL query, called `Expand.sql`[2].

```
SELECT   $(H.lo + I_C.idx * H.sp)$
FROM     $H, I_C, I_A$
WHERE    $I_C.idx \leq H.ct$ and $I_A.idx \leq H.avg$;
```

Here, H is the histogram stored as a relation and I_A, I_C are auxiliary relations, each with a single attribute idx. Relation I_A (resp., I_C) contains the integers $1, 2, .., A$ (resp., $1, 2, .., C$), where A (resp., C) is the largest *average frequency* (resp., *count*) in the buckets of H. Essentially, this query uses I_C to generate the positions of values within each bucket and then uses the *low* and *spread* values of the bucket to compute each of the approximate values, under the uniform spread assumption. Then, it uses I_A to replicate each value based on its frequency.

The approximate distribution captured by the above histogram looks as follows.

Quantity	Data Distribution Element				
Approx. Value	10	60	75	90	100
Approx. Frequency	100	70	70	70	2000

Histograms can also be built on multiple attributes together, by partitioning the joint distribution of the attributes into multi-dimensional buckets and using extensions of the uniform frequency and spread assumptions. It is also possible to *combine* two histograms on different sets of attributes to obtain a single histogram on the union of those two sets by making the *attribute value independence assumption*. Details on multi-dimensional histograms are given elsewhere [16]. In practice, there may be several one- or multi-dimensional histograms maintained on the attributes of a relation R. For simplicity of presentation, we assume that there is a single (multi-dimensional) histogram maintained on the full set of attributes of each relation.

Given the mechanisms of approximation within a histogram, it is clear that the accuracy of the approximation is determined by which attribute values are grouped together into each bucket. Several partitioning rules have been proposed for this purpose. For example, in an *equi-width* histogram, all buckets are assigned value ranges of equal width; in an *equi-depth* histogram, all buckets are assigned the same total number of tuples. In earlier work, we have introduced several new classes of histograms and identified a particular class of histograms, that we call *MaxDiff(V,A)*, which performs the best in estimating the selectivities of most kinds of queries. In a β-bucket *MaxDiff(V,A)* histogram, there is a bucket boundary between two successive attribute values if the difference between the *areas* of these values is one of the $\beta - 1$ largest such differences. By avoiding grouping dissimilar frequencies or spreads, the MaxDiff(V,A) histogram ensures that the uniform frequency and spread assumptions do not cause much errors. As an illustration, consider the histogram presented in Example 4.1, which is a MaxDiff(V,A) histogram. Note that it clearly separates the value 10 (skewed attribute value) and the value 100 (skewed frequency) from others.

[2]It is straightforward to generalize it so that it works with a multi-dimensional histogram, but it becomes quite complex without offering any new insight, so we do not present it.

5 Query Processing Using Histograms

In this section, we develop a histogram-based solution to approximate query answering.

First, we define the notion of a *valid approximate answer* to a query using histograms. Let *ApproxRel(H)* be the approximate relation corresponding to a histogram H on relation R (Section 4). The following definition captures the intuition behind an approximate query answer based on histograms.

Definition 5.1 Consider a query Q operating on relations $R_1..R_n$, and let $H_1..H_n$ be corresponding histograms. The *valid approximate answer* for Q and $\{H_i\}$ is the result of executing Q on ApproxRel(H_i) in place of R_i, for $1 \leq i \leq n$.

Next, we present two different ways to provide valid approximate answers.

5.1 Naive Approach

This approach is a direct application of Definition 5.1 and involves two steps: first, compute the approximate relations of all the histograms on the relations in the query (using the `Expand.sql` query given in the previous section); next, execute the query Q on these relations. This approach is clearly impractical because ApproxRel(H_i) may have as many tuples as R_i itself.

5.2 Efficient Approach

First, we formally define a *valid translation* of a query.

Definition 5.2 Consider a query Q operating on relations $R_1..R_n$, and let $H_1..H_n$ be corresponding histograms. A query Q' on these histograms is a *valid translation* of Q if the result of Q' is a histogram whose corresponding approximate relation is identical to the *valid approximate answer* for Q and $\{H_i\}$.

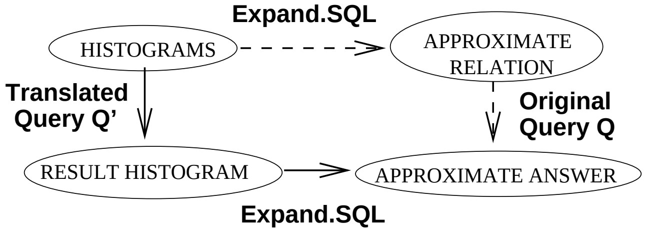

Figure 5: Valid Query Translation

Definition 5.2 is illustrated in the transition diagram of Figure 5. Essentially, Q' is a valid translation of Q when both paths from the HISTOGRAMS node to the APPROXIMATE ANSWERS node generate the same answer. The dashed path in Figure 5 corresponds to the above naive application of Definition 5.1 to obtain a valid query answer. The solid path in the figure, however, suggests the following, much more efficient approach to obtain the same result:

1. Obtain a valid translation Q' of Q
2. Execute Q' on $\{H_i\}$ to obtain a result histogram H_{res}
3. Compute ApproxRel(H_{res}) using `Expand.sql`

Since most of the query processing takes place on small histogram relations, this approach is clearly very efficient.

The last two steps above are straightforward. The rest of this subsection concentrates on the first step and provides valid translations for various query classes. We consider aggregate and non-aggregate SQL queries containing just *Select*, *From*, and *Where* clauses, but no *nesting*, *Group-By*, or *Having* clauses. These features can be added in a straightforward manner.

5.2.1 Non-Aggregate Queries

These queries are equivalent to relational algebra expressions involving just *selection*, *projection*, and *join* operations. A query Q in this category is translated as follows:

1. Construct an operator tree T of *select*, *project*, and *join* operations that is equivalent to Q.
2. Replace all the base relations in T by their corresponding histograms (that is, histogram relations as described in Section 4 to obtain another tree T'.
3. Starting from the bottom of T', translate each operator into an SQL query that takes the histograms from the operator's children and generates another histogram as output.

Note that, in general, there are many algebraic expressions that may be chosen in step 1 of the translation process, each giving a different valid translation. Although they may differ in cost, these costs are so low that there is no real need to optimize among the algebraic expressions.

The key contribution in the above process is step 3, as it involves valid translations of individual operators, which although not very complex are not completely straightforward either. They are described in detail below. For simplicity, we only deal with one-dimensional histograms (the proof of validity and extensions to multi-dimensional histograms are straightforward and are given elsewhere [8].

1. Equality Selection ($\sigma_{A=c}$): Equality selection is translated into the following query $Q_=$:

```
SELECT   $c, c, 1, avg$
FROM     $H$
WHERE    $(c \geq lo)$ and $(c \leq hi)$ and
         $(mod(c - lo, sp) = 0)$;
```

2. Range Selection ($\sigma_{A\leq c}$): Range selection is translated into the query $Q_\sigma = Q_a \cup Q_b$, where Q_a and Q_b are given below:

```
$Q_a$ :  SELECT   *
         FROM     $H$
         WHERE    $hi \leq c$;

$Q_b$ :  SELECT   $lo, lo + sp * \lfloor \frac{c-lo}{sp} \rfloor *$
                  $count, \lfloor \frac{c-lo}{sp} \rfloor * count, avg$
         FROM     $H$
         WHERE    $(lo \leq c)$ and $(hi > c)$;
```

3. Projection (π_A): Projection with duplicate elimination is translated into the following query Q_π:

```
SELECT   $lo, hi, count, 1$
FROM     $H$;
```

Projection with no duplicate elimination is just the identity query (i.e., selecting all tuples from the histogram relation with no changes).

4. Equi-Joins ($R_1 \bowtie_{R_1.A=R_2.B} R_2$): Let H_i be the histogram on the joining attribute of R_i, and N_i be the largest count in the buckets of H_i. Join is translated into a sequence of two queries, $Q1_{\bowtie}$ and $Q2_{\bowtie}$[3]. Query $Q1_{\bowtie}$ computes the frequency distribution of the approximate join result by joining the approximate frequency distributions of H_1 and H_2. It assumes the existence of two auxiliary relations of integers I_{N_1} and I_{N_2} defined in the same fashion as I_C described earlier.

```
SELECT   $(H_1.lo + I_{N_1}.idx * H_1.sp)$ as $v$, $H_1.lo$ as $lo_1$,
         $H_2.lo$ as $lo_2$, $H_1.avg * H_2.avg$ as $navg$
FROM     $H_1, H_2, I_{N_1}, I_{N_2}$
WHERE    $(H_1.lo + I_{N_1}.idx * H_1.sp =$
         $H_2.lo + I_{N_2}.idx * H_2.sp)$ and
         $(I_{N_1}.idx \leq H_1.count)$ and
         $(I_{N_2}.idx \leq H_2.count)$;
```

Query $Q2_{\bowtie}$ converts the result of query $Q1_{\bowtie}$ (say, $Q1R$) into a histogram by appropriate grouping.

```
SELECT     $min(v), max(v), count(*), navg$
FROM       $Q1R$
Group By   $lo_1, lo_2, navg$;
```

Example 5.1 Consider the following SQL query:

```
SELECT   $R2.B$
FROM     $R1, R_2$
WHERE    $R1.A \leq 10$ and $R1.B = R2.B$;
```

An equivalent operator tree and the corresponding translation result (a histogram query sequence depicted as nodes in a tree) are shown in Figure 6.

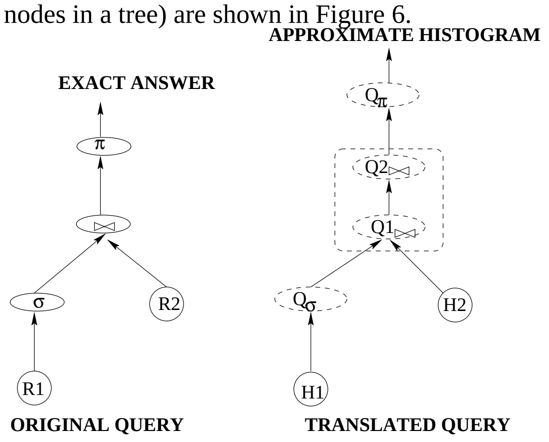

Figure 6: Example Query Translation

5.2.2 Aggregate Queries

In general, an aggregate query Q_{agg} is equivalent to an aggregate computation over some of the attributes in the result of a non-aggregate query Q. Hence, a valid translation for Q_{agg} consists of a valid translation for Q producing a histogram H followed by an aggregate-specific SQL query on H computing a single bucket histogram containing the aggregate value. These queries are given in Table 1 for the most common aggregate operators. Here, $bsum$ is the sum of all the attribute values in a bucket as many times as each appears, i.e., $bsum = avg * count * (lo + \frac{sp*(count-1)}{2})$.

5.2.3 Computational Complexity of Translated Queries

It can be easily seen that the queries for *selections* and *projections* access just β tuples, where β is the number of buckets in the histogram. This is usually insignificant compared to the number of tuples in the relations. For joins, the translated queries access $u_1 + u_2 + \beta_1 + \beta_2$ tuples, where u_i is the number of distinct attribute values in H_i and β_i is the number of buckets in H_i. The total number of operations is also proportional to this term because this query can be best run using Sort-Merge joins by always storing the histograms and the auxiliary relations in sorted order. The complexity is significantly smaller than the cost of running the original query because u_i is typically much smaller than the cardinality of the corresponding relation.

6 Experiments

We have conducted an extensive set of experiments using AQUA to study the effectiveness of various statistical techniques in providing approximate query answers. Our experiments involve different data sets and queries with set-valued as well as aggregate results. First, we present the testbed.

6.1 Testbed

6.1.1 Approximation Techniques

We have used different classes of histograms to approximate the data. They include the MaxDiff(V,A), Equi-Width, and EquiDepth classes. We call the corresponding approximate answering techniques *MaxDiff*, *Equi-Width*, and *EquiDepth* respectively. We have also studied the traditional uniformity assumption over the entire data, which is equivalent to a histogram with a single bucket. We call this technique *Trivial*. The final technique, which we call *Sampling*, uses sampling to provide approximate answers. Here, a set of samples is

[3]In our implementation, we make this scheme more efficient by running another simple query in the beginning to identify overlapping buckets in the histograms and then executing $Q1_{\bowtie}$ and $Q2_{\bowtie}$ for each pair of overlapping buckets.

distinct COUNT	SUM	AVG	MAX	MIN
SELECT SUM$(count)$ FROM H;	SELECT SUM$(bsum)$ FROM H;	SELECT $\frac{\texttt{SUM}(bsum)}{\texttt{SUM}(count)}$ FROM H;	SELECT MAX(hi) FROM H;	SELECT MIN(lo) FROM H;

Table 1: Queries to Compute Aggregate Values from Histograms

collected on each relation in the database and the submitted query is executed on the sample relations, with appropriate scaling of the final result.

In order to ensure a fair comparison among these techniques, we have allocated the same amount of space to each one. We have computed the number of buckets and samples corresponding to a space of s bytes as follows. Consider a relation with d integer attributes. Since each bucket in a d-dimensional histogram stores $(3 * d + 1)$ numbers (Section 4), the number of buckets is $\frac{s}{12*d+4}$ (assuming 4 bytes per number). Similarly, since each sampled tuple contains d numbers, the corresponding sample set contains $\frac{s}{4*d}$ tuples.

6.1.2 Data Sets

We have primarily used a synthetic database that we have created. We have also experimented with the TPC-D benchmark database [18], generated at scale factor of 0.6. Due to lack of space, we focus on our synthetic data, which offers more insights into the performance of various techniques than the TPC-D data, which has mostly uniform and independent attributes. The synthetic data consists of two relations R_1 and R_2; R_1 contains two numerical attributes (A, B), and R_2 consists of a single numerical attribute (A); having additional attributes does not impact our study. The distribution of data in these attributes is described next.

Value Domain: The attribute values were generated from a combination of *Zipf* distributions [21]. The details of all such combinations are given elsewhere [17]; here we describe the two that we chose to present in this paper. The *Cusp-Max* distribution consists of increasing spreads (distances between successive values) followed by decreasing spreads, with the spreads taken from a Zipf distribution. The skew in the spreads is controlled via the z parameter (higher z implies higher skew). The *Uniform* distribution consists of equally spaced values.

Frequency Domain: The frequencies of the different attribute value combinations were also generated based on Zipf distributions with different levels of skew.

The set of values that we have experimented with for all data set parameters is given in Table 2.

Parameter	Values
Value Skew for *Cusp-Max* (z_v)	$0.2 - 3$
Frequency Skew (z_f)	$0 - 3$
Num. Distinct Values per Dimension (u)	$10 - 10K$
Num. Attribute Combinations (U)	$500 - 50K$
Num. Tuples in the Relation (T)	$50K - 500K$

Table 2: Synthetic Data Set Parameters

6.1.3 Queries

Non-Aggregate Queries: The SQL queries used are listed in Table 3. Here, we summarize their key characteristics.

- *Range Queries:* These queries contain a single two-dimensional *range selection* predicate on $R1$. In each experiment, we have used 100 queries with randomly generated values for a and b. These values were chosen from a subset of the entire column range in order to vary the average selectivity of the queries.
- *Projection Queries:* These queries simply project a relation onto one of its attributes, with duplicate elimination.
- *SJ Queries:* These queries contain both *join* and *selection* operations.

Aggregate Queries: In addition to these set-valued queries, we have also examined versions of them that produce common aggregations on their first column. Due to lack of space, we only present the results for *average*.

Error Metrics: For all queries, we have used the *MAC* error measure introduced in Section 3. In the graphs for aggregate queries, we plot the percentage relative error, i.e., $\frac{MAC(=actual-estimate)}{actual} * 100$.

6.2 Experimental Results

Here, we first present the experimental results for different query sets and then provide the explanation for them.

6.2.1 Range Queries

For this set of experiments, the number of tuples in $R1$ was $200K$ and the average selectivity of the queries was approximately 0.15.

Non-aggregate Queries:

Range Query		Projection Query		SJ Query	
`SELECT`	$R_1.A, R1.B$	`SELECT`	`distinct` $R1.A$	`SELECT`	$R_1.A$
`FROM`	R_1	`FROM`	R_1;	`FROM`	R_1, R_2
`WHERE`	$(R_1.A \leq a)$ `and` $(R1.B \leq b)$			`WHERE`	$(R_1.A = R_2.A)$ `and` $(R1.B \leq c)$;

Table 3: Non-Aggregate Queries in the Experiment Testbed

Effect of Space: Here, we have fixed the frequency skew of both attributes $R1.A$ and $R2.A$ at $z_f = 0.86$ (which roughly corresponds to the "80-20" rule) and have chosen the value domain as *Uniform*. Figure 7 shows the errors due to various techniques (in log scale on the y-axis) as the space is varied (in log scale on the x-axis). The error for *Trivial* is not given because it is very high (above 4×10^6) and falls out of the range depicted. As expected, the performance of the various techniques improves with increasing space.

Effect of Frequency Skew: For this experiment, we have fixed the space at 400 bytes and have chosen again a *Uniform* value domain. Figure 8 depicts frequency skew (z value) on the X-axis and *MAC* errors (in log scale) on the y-axis.

Aggregate Queries: The errors for *average* are given for varying amounts of space in Figure 9.

Note that *MaxDiff* performs very well under all circumstances.

6.2.2 Projections (with no aggregates)

In Figure 10, we compare the performance of various techniques for the *Cusp-Max* value distribution. The frequency skew is 0 and the values of $R.A1$ range from 0 to 10000. Once again, *MaxDiff* performs the best and *Sampling* performs very poorly.

6.2.3 SJ Queries (with no aggregates)

Here, R_1 and R_2 contain 100000 and 1000 tuples, respectively, and have 500 distinct values each. The range predicate has a selectivity of 10%. The value distributions of R_1 and R_2 are both *Cusp-Max*, with z_v of 0.2 and 1, respectively. The frequency distributions of both relations have a skew of $z_f = 1.5$. The errors in estimating the result set are given in Figures 11 and 12 as functions of space and frequency skew, respectively. Note that *MaxDiff* performs very well for the SJ-queries as well.

6.2.4 Explanation of Results

All of the above results demonstrate a similar pattern in the relative behavior of the techniques studied. In particular, *MaxDiff* performs the best in most circumstances. The reasons are that a) it approximates the *entire* data distribution, and b) it captures the more skewed attribute values with high accuracy using a constant amount of space (1 bucket per such value). In contrast, *Sampling* captures a fraction of the given set precisely and misses the remaining parts completely. Furthermore, it allocates disproportionate amounts of space to the high frequency values. Since our *selection* queries cover the *entire value domain*, many of them contain the low-skewed regions that may not be captured (or even approximated) at all in *Sampling*, resulting in a high error. The problem gets worse for SJ-queries because join of two samples often contains few or no tuples [1]. Some of the sampling-related problems have been addressed elsewhere in a different context [4, 1]. We are currently incorporating these optimizations in our work.

As for the effect of skew, errors due to *Sampling* increase with skew because it allocates more and more samples for the high frequency values. This degrades performance for queries on the remaining regions. On the other hand, *EquiWidth* and *EquiDepth* perform poorly because they do not consider frequency skews much in forming the buckets, unlike *MaxDiff*. Interestingly, *MaxDiff* performs best for extreme values of skew (0 & 3). The reasons are as follows: at high skew values, there are very few "important" values in the relation that need to be captured and histograms are able to do that using a small number of buckets: at low skew values, almost all frequencies are identical and even one or two buckets are enough. For medium values of skew there are sufficient number of distinct frequencies, so that the histogram needs more buckets to accurately approximate the distribution.

6.2.5 Times Taken by Different Approaches

We have measured the times taken by various techniques in answering the SJ queries on a SUN SPARC machine with 250MB of memory and 10GB of disk space. Evaluation of the exact answer for these queries took around 248 seconds (averaged over 30 runs). Table 4 lists the times for different values of space. Note

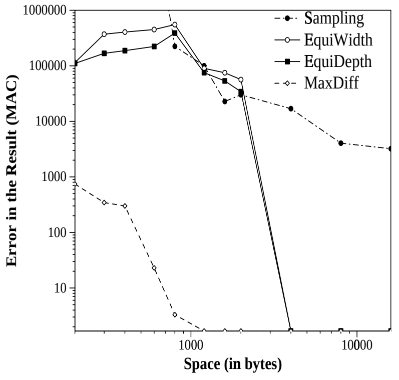

Figure 7: Effect of Space on *MAC* (Range Queries)

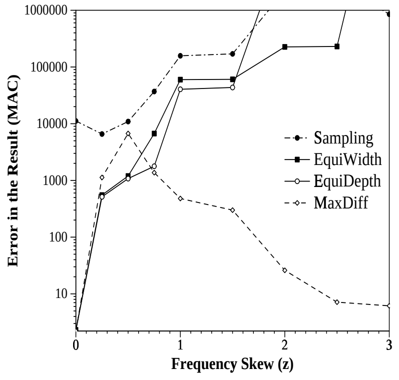

Figure 8: Effect of Frequency Skew on *MAC* (Range Queries)

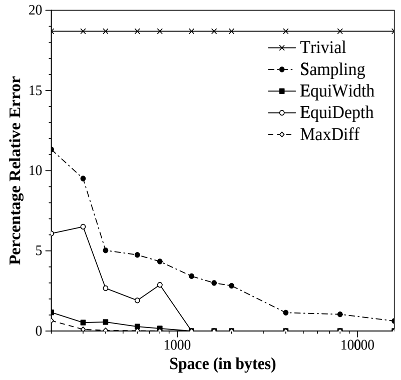

Figure 9: Effect of Space on Estimating Averages (Range Queries)

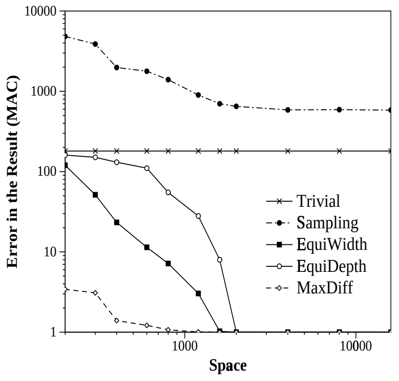

Figure 10: Effect of Space on *MAC* (Projections)

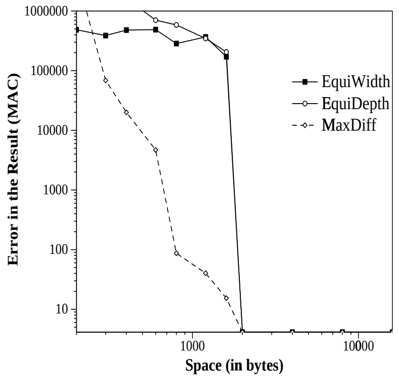

Figure 11: Effect of Space on *MAC* (SJ Queries)

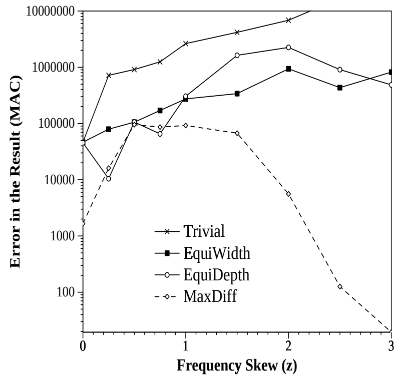

Figure 12: Effect of Frequency Skew on *MAC* (SJ Queries)

Technique	Space (in bytes)					
	50	100	500	1000	2000	4000
Sampling	0.10	0.18	0.72	2.60	5.30	10.30
MaxDiff	0.56	0.82	1.32	3.20	6.10	11.19

Table 4: Times Taken for Answering SJ-Queries (sec)

that *MaxDiff* takes slightly more time than *Sampling*, as join queries on histograms require as many operations as the number of distinct values in the join attributes. Nevertheless, the times are still very small and are insignificant compared to the actual time of execution.

7 Conclusions

Approximate query answering is likely to become an essential tool in applications demanding fast responses, such as on-line decision support systems and interactive data visualization tools. However, the work in this area so far has been limited in its scope. First, it has only considered aggregate queries and has not dealt with queries that return multisets of tuples. Second, sampling has been essentially the only technique used.

In this paper, we have attempted to increase the scope of approximate query answering as follows:

- We have developed a numerical measure (*MAC*) to quantify the quality of an approximate answer to set-valued query. We have demonstrated that this measure is intuitive, reduces to well-known metrics in simple cases, and works well for most instances of answer-sets.
- We have proposed histogram-based techniques for providing approximate answers to general as well as aggregate queries. In this regard, we have devised a histogram algebra for transforming queries on regular relations to queries on histograms.

We believe that the error measure and the histogram algebra form a reasonable first step towards systematically providing approximate query answers for general queries. Furthermore, the results of our experiments indicate that *MaxDiff* histograms answer most kinds of queries very accurately, like in selectivity estimation.

Acknowledgements: The authors would like to thank S. Muthukrishnan for help with the exact algorithm for identifying the minimum cost edge cover.

References

[1] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswarmy. Join synopses for improving approximate query answers. *Proc. of ACM SIGMOD Conf*, 1999.

[2] P. Buneman, S. B. Davidson, and A. Watters. A semantics for complex objects and approximate queries. In *Proc. 7th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, pages 305–314, April 1988.

[3] C. M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. *Proc. of ACM SIGMOD Conf*, pages 161–172, May 1994.

[4] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. *Proc. of ACM SIGMOD Conf*, 1998.

[5] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *Proc. of the 23rd Int. Conf. on Very Large Databases*, August 1997.

[6] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Proc. ACM SIGMOD Conference on the Management of Data*, pages 171–182, Tucson, AZ, June 1997.

[7] Huttenlocher. Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9), September 1993.

[8] Y. Ioannidis and V. Poosala. Histogram-based techniques for approximating set-valued queries. Technical report, Bell Labs, 1998. Contact poosala@bell-labs.com for a copy.

[9] R. P. Kooi. *The optimization of queries in relational databases*. PhD thesis, Case Western Reserve University, Sept 1980.

[10] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. *Proc. of ACM SIGMOD Conf*, pages 1–11, May 1990.

[11] A. Motro. Query generalization: A method for interpreting null answers. In L. Kerschberg, editor, *Expert Database Systems, Proceedings from the First International Workshop*, pages 597–616. Benjamin/Cummings, Inc., Menlo Park, CA, 1986.

[12] G. Ozsoyoglu, K. Du, S. Guruswamy, and W.-C. Hou. Processing real-time non-aggregate queries with time-constratings in CASE-DB. In *Proc. 8th International Conference on Data Engineering*, pages 140–147, Tempe, AZ, February 1992.

[13] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. *Proc. of ACM SIGMOD Conf*, pages 256–276, 1984.

[14] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. *International working conference on scientific and statistical database management*, 1999.

[15] V. Poosala and V. Ganti. Fast approximate query answering using precomputed statistics. *Proc. of IEEE Conf. on Data Engineering*, 1999.

[16] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. *Proc. of the 23rd Int. Conf. on Very Large Databases*, August 1997.

[17] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. *Proc. of ACM SIGMOD Conf*, pages 294–305, June 1996.

[18] Transaction processing performance council (TPC). *TPC-D Benchmark Manual*, 1996.

[19] J. S. Vitter, M. Wang, and B. R. Iyer. Data cube approximation and histograms via wavelets. *CIKM*, pages 96–104, 1998.

[20] S. Vrbsky and J. W. S. Liu. APPROXIMATE: A query processor that produces monotonically improving approximate answers. *IEEE Trans. on Knowledge and Data Engineering*, 5(6), December 1993.

[21] G. K. Zipf. *Human behaviour and the principle of least effort*. Addison-Wesley, Reading, MA, 1949.