Approximate Query Answering using Histograms [0] *Copyright 1999 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.*

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Viswanath Poosala	Venkatesh Ganti	Yannis E. Ioannidis
poosala@lucent.com	vganti@cs.wisc.edu	yannis@cs.wisc.edu
Bell Laboratories	Department of Computer Sciences	Department of Informatics
Lucent Technologies	Univ. of Wisconsin-Madison	Univ. of Athens
Murray Hill, NJ, USA	Madison, WI, USA	Athens, Greece

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. In this article, we explore the use of precomputed histograms *for approximate answering of aggregate queries. Histograms are used by most database systems for selectivity estimation within their optimizers. However, the use of histograms for approximate query answering raises several novel issues, which are addressed in this article. We present a* histogram algebra *for efficiently executing complex SQL queries on histograms within a DBMS without requiring any changes to the DBMS internals. We enhance histograms to estimate the quality of the approximate answers. Finally, we present an efficient technique for selecting a provably* near-optimal *set of histograms on the data cube, which minimizes the space needed when an upper bound on errors is given.*

1 Introduction

The users of decision support applications pose very complex queries to Database Management Systems (DBMSs), which take a long time to execute. 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 their complete execution if the approximate answer indicated something interesting.

*[

Several techniques have been proposed for providing approximate answers using statistical summaries of the data, such as samples, histograms, and wavelets. In this article we focus on histograms, which have the advantages of being present in almost every commercial DBMS and being reasonably accurate (for selectivity estimation). There are also two different ways to present approximate answers: the *online aggregation* approach constantly refines the answer using larger and larger sets of statistics of the data until the accurate answer is obtained [8], whereas the *precomputation* approach presents a small number of discrete approximate answers (typically, just one) by using precomputed summaries of the data [1]. The precomputation approach has the advantage of being faster in providing an answer because only the small summary data has to be processed at run-time; on the other hand online aggregation has the flexibility of refining the answers. However, it is possible to deploy a precomputation approach without requiring any changes to the DBMS, while online aggregation requires new query processing and data access strategies to be implemented in the DBMS. Based on these trade-offs, we use precomputed histograms in our work[1].

In this paper, we investigate the above approach to approximate query answering and present two different ways of using histograms for this purpose. Though histograms have been widely used in databases, their usage has been mostly restricted to selectivity estimation [15, 9, 19, 12, 14]. The use of histograms for approximate query answering brings up several novel issues to fore, which form the main focus of this article. Our contributions are summarized below.

- **Efficient Query Execution:** We propose 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 define a *histogram algebra* that can be used for this translation.

- **Quality of Answers:** In order to provide any confidence to the user in using an approximate answer, a measure of the quality of the answer must also be provided. Unlike sampling-based techniques which often provide a confidence measure for this purpose, traditional histogram-based techniques do not offer any error measures. Hence, we enhance histograms to provide quality guarantees on the approximate answers.

- **Histogram Selection:** Given the limited resources for storing summaries, it is important to choose histograms that result in the most accurate answers to queries while using small amounts of space. Presumably, a common situation in an approximate query answering system is where the user specifies a bound on the errors in the answers and demands minimal space usage. We provide an efficient greedy technique for selecting a near-optimal set of histograms on the data cube when an error bound is given.

Due to space limitations, we omit our experimental study from this article, and refer the reader to other papers for those

This work constitutes a part of our efforts to build an efficient data analysis system called Aqua [1]. In this system, we store the statistics (histograms and samples) as relations in the DBMS and rewrite the user query posed on the original relations as a query on the statistics relations. The rewritten query is then submitted to the DBMS for execution. This middleware architecture, coupled with the provision of standard ODBC interfaces, enables Aqua to be easily deployed between almost any front-end querying system and backend DBMS. More details about the Aqua system can be found in [1].

Next, we provide a background on histogram techniques.

[1] It may also be possible to perform online aggregation using histograms, however that is outside the scope of this article.

2 Histograms

In this section, we summarize standard histogram-based techniques for approximating the data in a database [19, 18]. First, we present some useful definitions.

The *value set* $\mathcal{V}_i$ of attribute X_i 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.

A *histogram* on an attribute X is constructed by using a *partitioning rule* to partition its data distribution into β (≥ 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 [19], 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*). Example 1 illustrates the above concepts.

Example 1: The following table shows how each parameter defined above is instantiated for a hypothetical attribute.

Quantity	**Data Distribution Element**					
Value	10	60	70	80	85	100
Frequency	100	120	600	200	80	200
Spread	50	10	10	5	15	1
Area	5000	1200	6000	1000	1200	200

Consider a 4-bucket histogram on this attribute with the following bucketization of attribute values: $\{ 10 \}$, $\{ 60 \}$, $\{ 70 \}$, $\{ 80, 85, 100 \}$. The approximate distribution captured by this histogram looks as follows.

Quantity	**Data Distribution Element**					
Approx. Value	10	60	70	80	*90*	100
Approx. Frequency	100	125	600	160	160	160

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.

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*. All of these details are given in [18]. In practice, there may be several one- and/or multi-dimensional histograms on a relation R. For simplicity of presentation, we assume in the rest of the paper that there is a single (multi-dimensional) histogram on a relation computed under the above assumption.

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 *V-Optima(V,A)*, which performs the best in estimating the selectivities of most kinds of queries. In a *V-Optimal(V,A)* histogram, buckets are formed such that the sum of the weighted variances of the areas within the buckets is minimized (the weights being the number of values in that bucket). Recall that *area* captures both the frequency and value domains. By trying to group together similar frequencies and spreads, the V-Optimal(V,A) histogram ensures that the uniform frequency and spread assumptions do not cause much errors.

With respect to storage, 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 set of columns: the average spreads along each dimension ($sp_i = \frac{hi_i - lo_i}{u_i - 1}$) and the average frequency for the bucket ($avg = \frac{tot}{u_1 u_2 \ldots u_d}$). In the rest of the paper, the term *histogram* refers to the histogram relation described here. For illustration, we present the histogram relation for the histogram given in Example 1:

lo	hi	$count$	tot	sp	avg
10	10	1	100	0	100
50	60	2	250	10	125
70	70	1	600	0	600
80	100	3	480	10	160

3 Approximate Query Answering using Histograms

There are essentially two different ways to use histograms for approximate query answering, roughly corresponding to the relational and multi-dimensional OLAP approaches. They are described below.

Relational Approximation: In this approach, histograms are used to approximate the data in the relation (as explained in the previous section). Queries on the data are answered based on the approximate relations captured by the histograms. The key issue in relational approximation is the efficient execution of queries. This is addressed in Section 4.

Data Cube Approximation: In this approach, histograms are used to approximate the OLAP *data cube* and aggregate queries on the data cube are answered from the "approximate data cube". A data cube is essentially a hierarchy of multi-dimensional *sub-cubes*, where each sub-cube describes the aggregate distribution of data in a subset of dimensions [6]. The axes of the sub-cube contain the distinct values in each of the *dimensional attributes*; and the cells contain the aggregate (sum, max, min, etc) value of one of the *measured attributes* in the relation. An *approximate sub-cube* has the same structure as the original sub-cube, but with approximate values on the cells and the axes. In particular, we obtain the approximation by building a histogram on the multi-dimensional distribution represented by the sub-cube, i.e., *the dimensions form the value domains and the cell values are treated as their frequencies*.

We illustrate this using the following example.

Example 2: Consider the `LINEITEM` table in the TPC-D database and three of its attributes `Supplier id (S)`, `Customer id (C)`, and `Part id (P)`. Figure 1 shows the tuples in an example `LINEITEM` table. Figures 2 and 3 show the sub-cubes corresponding to applying `SUM` aggregate on { `P,S` } and { `P` } respectively. In Figure 4, a trivial approximate sub-cube for { `P,S` } is shown, which is obtained by replacing all the measure values by their average (= total (54)/count (6)), i.e., using a single bucket histogram.

Note that any query on the original sub-cube { `P,S` } can also be answered, albeit inaccurately, from the approximate sub-cube. In order to answer all queries on the data cube, one needs to approximate the entire data cube while using small amount of space. This is addressed in Section 6.

P	S	C	Q
100	200	300	1
100	200	301	7
100	201	300	3
100	201	301	9
101	200	302	11
101	202	300	23

Figure 1: { P, S, C }

	S		
P	200	201	202
100	8	12	0
101	11	0	23

Figure 2: { P, S }

P	
100	101
20	34

Figure 3: { P }

	S		
P	200	201	202
100	9	9	9
101	9	9	9

Figure 4: Approx { P, S }

4 Query Execution on Histograms

In this section, we develop techniques for automatically translating SQL queries on original relations into SQL queries on histogram relations. These techniques are used in the relational approximation approach described above.

First, we define the notion of providing *valid approximate answers* to a query using histograms. Let *ApproxRel(H)* be the approximate relation corresponding to a histogram H on relation R (Section 2). Then, we believe that the following definition captures the intuition behind an approximate query answer based on histograms.

Definition 1: Consider a query Q operating on relations $R_1..R_n$, and let H_i be the histogram on R_i. 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 $1 \leq i \leq n$.

An obvious way to derive the valid approximate answer is as follows: first, compute the approximate relations of all the histograms on the relations in the query; next, execute the query Q on these relations. 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 $I_C.idx \leq H.ct$ & $I_A.idx \leq H.avg$;

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.

However, this approach is inefficient because ApproxRel(H_i) may have as many tuples as R_i itself, thus defeating the whole purpose of approximate query answering. Hence, we describe a far more efficient approach for computing valid approximate answers, which executes directly on histograms:

1. Obtain a *valid translation* Q' of Q, which would at the end of these three steps yield a *valid approximate answer* (described next).

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.

[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.

4.1 Translations for 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 to obtain another tree T'.
3. Starting from the bottom of T', translate each operator into an SQL query that takes one or two histograms from the operator's children and generates another histogram as output.

The translations for various query operators are described below.

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

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

- **Range Selection** ($\sigma_{A\leq c}$): Range selection is translated into the following query Q_σ:

SELECT *		SELECT $lo, lo + sp * \lfloor \frac{c-lo}{sp} \rfloor, \lfloor \frac{c-lo}{sp} \rfloor, avg$
FROM H	$\cup$	FROM H
WHERE $hi \leq c$;		WHERE $(lo \leq c)$ & $(hi > c)$;

- **Projection** (π_A): Assuming duplicate elimination, projection is translated into the following query Q_π:

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

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

- **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]. The first 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, $S.lo$ as lo_2, $H_1.avg * H_2.avg$ as $navg$
  FROM $H_1, H_2, I_{N_1}, I_{N_2}$
  WHERE $(H_1.lo + I_{N_1}.idx * H_1.sp = H_2.lo + I_{N_2}.idx * H_2.sp)$ &
  $(I_{N_1}.idx \leq H_1.count)$ & $(I_{N_2}.idx \leq H_2.count)$;

  The second query ($Q2_\bowtie$) converts the result of query Q (say, $Q1R$) into a histogram by appropriate grouping.

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

[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

4.2 Aggregate Queries

In general, an aggregate query Q_{agg} can be viewed as computing aggregates 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 values in a bucket, i.e., $(avg * count * (lo + \frac{sp*(count-1)}{2}))$.

It has been shown that the techniques presented here result in orders of magnitude faster execution than obtaining the exact answer [10].

5 Quality of Answers

An important requirement of any system providing approximate answers is a quality measure on the answers and a way to estimate this measure when the actual answer is not available. Histograms can be supplemented with additional information in order to estimate the error in an approximate answer. Here, we describe the estimation of errors for data cube approximation; similar approach can be taken for relational approximation.

Consider a group-by query applying an aggregate operator on a sub-cube with dimensions $\mathcal{S}$. The error in estimating the result over a single group in the answer is the absolute difference between the actual and approximate aggregate values over that group. And, when the actual and approximate answers have the same set of groups, the error in answering the entire query is the sum of errors over all the groups in the result[4]. We describe a simple way to compute an *upper bound* on this error for the *sum* operator below. Similar techniques can be derived for other aggregate operations as well.

Let $\{ b_1, .., b_n \}$ be the set of buckets in the histogram used in answering the query. With each bucket b_i, we also maintain the maximum difference (m_i) between the actual and the average measured values aggregated on $\mathcal{S}$ in that bucket. Then, the maximum error contributed by bucket b_i to the query is simply $m_i \times MIN(n_i, t_i - n_i)$, where n_i is the number of values in the bucket that satisfy the query predicate and t_i is total number of values in the bucket. The total error in answering the query is computed by adding the contributions from all the buckets.

6 Histogram Selection for Data Cube Approximation

Recall that the data cube consists of a set of sub-cubes. An obvious approach for approximating the data cube is to build a histogram on each sub-cube. We call this the *direct* approach. However, this requires considerable amount of space because the number of histograms needed is exponential in the number of dimensions of the fact relation. Fortunately, since a histogram on a sub-cube $\mathcal{S}$ also contains information on any of its sub-sets, say $\mathcal{P}$, we can use a histogram on $\mathcal{S}$ to provide an approximation for $\mathcal{P}$ by projecting it onto the attributes in $\mathcal{P}$. We call this the *slicing* approach.

[4]When the answers contain different sets of groups, one can use the *MAC error* that we devised in [10] for capturing the difference between any two sets of numbers - in this case the actual and approximate answers. However, we are still working on techniques for estimating this error when the actual answer is not known.

In view of the above alternatives for approximating a sub-cube, two natural questions arise: i) which histograms should be built and how much space should be allocated for each? and, ii) given a set of histograms computed on the data cube, which histogram should be used to answer a query on a sub-cube? First, we define two error metrics which are useful in answering these questions.

Definition 2: Let $\mathcal{S}'$ be an approximate sub-cube of the sub-cube $\mathcal{S}$. For a tuple t, let $A(t)$ be the measure value from $\mathcal{S}$, and $A'(t)$ its measure value from $\mathcal{S}'$. We define the *sub-cube error (E_S)* of $\mathcal{S}'$ as the average relative error between all the actual and the approximate measure values present in the sub-cube. That is[5],

$$E_S = \Sigma_{t \in \mathcal{S}} \frac{A(t) - A'(t)}{MAX(A(t), 1) \times |S|}$$

Definition 3: The *data cube error* (E) is the *maximum* of the sub-cube errors of all the sub-cubes in that data cube.

$$E = Max_{\mathcal{S} \in 2^{\mathcal{U}}}(E_S)$$

Then, for a given set of histograms on a data cube, we use the histogram that approximates a sub-cube with the least sub-cube error for answering queries on that sub-cube. The allocation of space among the histograms is addressed next.

We refer to a particular allocation of space among histograms as a *histogram configuration*. We developed techniques to identify histogram configurations that minimize the space needed when the error is upper-bounded.

Definition 4: For a given upper bound ϵ on the data cube error E (Definition 3), the *ϵ-optimal* configuration is the configuration that requires the least amount of space while resulting in an error of at most ϵ.

Earlier research on optimization problems similar in nature to this problem has shown many of them to be NP-Hard [13, 7]. Though the complexity of our specific problem has not yet been established, there is a strong likelihood that it may not have an efficient solution. However, we have shown in [16] that the identification of the *ϵ-optimal* configuration is upper-bounded by the *minimum weighted set cover* problem (MWSC). There, we adapted the standard greedy algorithm used for computing an approximation to the minimum weighted set cover to provide an efficient heuristic for the current problem. The configuration generated by this algorithm requires an amount of space that is bounded within a factor of that required by the optimal configuration. This algorithm is described next.

We use the notion of a data cube lattice [7], which contains a vertex for each sub-cube and an edge from u to v if v is a subset of u. Each node v in the lattice is associated with a "marking;" v is marked to indicate that the sub-cube associated with it is not yet approximated with the required accuracy ϵ. To start with, no histogram is built on any sub-cube, and all nodes are marked. Next, we define some terms used in the algorithm. Let u and v be two nodes in the lattice and $e = \vec{uv}$ be a directed edge in the lattice.

- *ApproxError(e, β)*: This is the sub-cube error in approximating the sub-cube v using a histogram of β buckets on u. It is calculated by actually building a histogram with β buckets on the sub-cube u and computing the sub-cube error (Definition 2).

- *weight(e)*: This is the least value of β such that ApproxError(e, β) is less than or equal to ϵ.

[5] we divide by the maximum of 1 and the actual value to handle the case where actual result is 0.

- *benefit(e)*: This is the number of *marked* children of u that can be answered within an error ϵ by allocating $weight(e)$ buckets to the histogram on u. That is, the additional number of sub-cubes that can be approximated within ϵ due to the allocation of additional buckets.

The algorithm is given below.

Algorithm 6.1: GREEDY(double ϵ)
/* ϵ is the bound on the data cube error */
begin
    for every edge e in the lattice **do**
        compute weight(e) by computing ApproxError(e, β) for increasing values of β.
    while (some of the vertices are still *marked*) **do**
    begin
        Pick an edge $e = \vec{uv}$ with the largest *benefit to weight ratio* and allocate a space of $weight(e)$ to u.
        Unmark all the children w of u with $ApproxError(uw, weights(e)) \leq \epsilon$.
        Update the benefits of the affected edges.
        /* Only those edges incident on v are affected by selecting e */
    end
end

---

7 Related Work

There has been significant amount of recent work on providing approximate answers to *aggregate queries* using precomputed statistics, such as, samples [1], histograms [17, 16], and wavelets [20]. Much of the work presented here on data cube approximation and histogram selection first appeared in [16]. Approximate answering of non-aggregate queries was addressed in [10], where a numerical measure for comparing set-valued answers was defined. Online aggregation [8], described earlier, constitutes another style of sampling-based approximate query answering wherein the answers are continuously refined till the exact answer is computed.

Histograms have been studied extensively for application in selectivity estimation in query optimizers [12, 14, 15]. In our earlier work, we have identified several novel classes of histograms to build on one or more attributes [19, 18] and also proposed techniques for their efficient computation [11] and incremental maintenance [5]. We recently extended histograms for selectivity estimation in spatial databases [2]. Much of the work presented in this article on query algebra for histograms has appeared in [10].

8 Conclusions

In this article, we have described various histogram-based techniques for providing approximate answers to aggregate queries. Histograms have been traditionally used for selectivity estimation. Their use for approximate query answering brings up several issues: efficient query processing, quality guarantees, and histogram selection being the three key issues addressed here. We have presented general solutions for all three issues, thus establishing the usability of histograms for this new problem.

References

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

[2] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. *Proc. of ACM SIGMOD Conf*, 1999.

[3] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA., 1989.

[4] M. Garey and D. Johnson. *Computers and intractability*. W. H. Freeman and Co., 1979.

[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. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tabs, and sub-totals. *Proc. of IEEE Conf. on Data Engineering*, pages 152–159, 1996.

[7] V. Harinarayanan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. *Proc. of ACM SIGMOD Conf*, pages 205–216, 1996.

[8] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *Proc. of ACM SIGMOD Conf*, 1996.

[9] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. *Proc. of ACM SIGMOD Conf*, pages 233–244, May 1995.

[10] Y. Ioannidis and V. Poosala. Histogram-based techniques for approximating set-valued query-answers. *Proc. of the 25th Int. Conf. on Very Large Databases*, 1999.

[11] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. *Proc. of ACM SIGMOD Conf*, 1998.

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

[13] S. Muthukrishnan, V. Poosala, and T. Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. *7th International Conference on Database Theory*, January, 1999.

[14] 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.

[15] V. Poosala. *Histogram-based estimation techniques in databases*. PhD thesis, Univ. of Wisconsin-Madison, 1997.

[16] 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.

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

[18] 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.

[19] 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.

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