Data-Streams and Histograms

Sudipto Guha* Nick Koudas† Kyuseok Shim‡

ABSTRACT

Histograms have been used widely to capture data distribution, to represent the data by a small number of step functions. Dynamic programming algorithms which provide optimal construction of these histograms exist, albeit running in quadratic time and linear space. In this paper we provide linear time construction of $1 + \epsilon$ approximation of optimal histograms, running in polylogarithmic space.

Our results extend to the context of data-streams, and in fact generalize to give $1 + \epsilon$ approximation of several problems in data-streams which require partitioning the index set into intervals. The only assumptions required are that the cost of an interval is monotonic under inclusion (larger interval has larger cost) and that the cost can be computed or approximated in small space. This exhibits a nice class of problems for which we can have near optimal data-stream algorithms.

1. INTRODUCTION

Histograms capture distribution statistics in a space efficient fashion. They have been designed to work well for numeric value domains, and have long been used to support cost-based query optimization [22, 11, 12, 25, 27, 26, 23, 14, 13, 15, 20, 17], approximate query answering [7, 2, 1, 29, 28, 24], data mining [16] and map simplification [3].

Query optimization is a problem of central interest to database systems. A database query is translated by a parser into a tree of physical database operators (denoting the dependencies between operators) that have to be executed and form the query answer. Each operator when executed incurs a cost (in terms of disk accesses) and the task of the query optimizer is to form the minimum cost execution plan. Histograms are used to estimate the cost of physical database operators in a query plan. Many operators of interest exist, the most common being select and join operators. A select operator commonly corresponds to an equality or a range predicate that has to be executed on the database and various works deal with the construction of good histograms for such operations [22, 11, 23, 15, 13, 20, 17]. The result of such estimation through the use of a histogram represents the approximate number of database tuples satisfying the predicate (commonly referred to as a selectivity estimate) and can determine whether a database index should be used (or constructed) to execute this operator. Join operators are of profound importance in databases and are costly (in the worst case quadratic) operations. Several proposals exist for the construction of good histograms to estimate the cost of join operations [12, 26, 14]. The estimates derived from such an estimation are used to determine the order with which multiple join operators should be applied in the database tables involved.

Histograms have also been used in approximate query answering systems, where the main objective is to provide a quick but approximate answer to a user query, providing error guarantees. The main principle behind the design of such systems is that for very large data sets on which execution of complex queries is time consuming, is much better to provide a fast approximate answer. This is very useful for quick and approximate analysis of large data sets [2]. Research has been conducted on the construction of histograms for this task [7, 1] as well as efficient approximations of datacubes [8] via histograms [28, 29, 24].

An additional application of histograms is data mining of large time series datasets. Histograms are an alternate way to compress time series information. Through the application of the minimum description length principle it is possible to quickly identify potentially interesting deviations in the underlying data distribution [16]. All the above applications share a common property, that the histograms are constructed on a dataset that is fully known in advance. Thus algorithms can construct good histograms by performing multiple passes on the data set.

The histogram approach is also useful in curve simplification, and specially in transmission of subsequent refinements of a distribution [3]. Fixing the initial transmission size, reduces the problem to approximating the distribution by a histogram. Subsequent transmissions carry more information. This has a similar flavor to approximating the data by wavelets, used in [28] amongst others. Haar wavelets are in fact very simple histograms, and this approach allows us to minimize some objective fit criterion, than storing the k highest coefficients of a wavelet decomposition.

*AT&T Research. Email: sudipto@research.att.com
†AT&T Research. Email: koudas@research.att.com
‡Computer Science Department and AITRC, KAIST. EMail: shim@cs.kaist.ac.kr

A data stream is an ordered sequence of points that can be read only once or a small number of times. Formally, a data stream is a sequence of points $x_1, \ldots, x_i, \ldots, x_n$ read in increasing order of the indices i. The performance of an algorithm that operates on data streams is measured by the number of passes the algorithm must make over the stream, when constrained in terms of available memory, in addition to the more conventional measures. Such algorithms are of great importance in many networking applications. Many data sources belonging to mission critical network components produce vast amounts of data on a stream, which require online analysis and querying. Such components include router link interfaces, router fault monitors and network aggregation points. Applications such as dynamic traffic configuration, fault identification and troubleshooting, query the data sources for properties of interest. For example, a monitor on a router link interface commonly requests for the total number of bytes through the interface within arbitrary windows of interest.

One of the first results in data streams was the result of Munro and Paterson [21], where they studied the space requirement of selection and sorting as a function of the number of passes over the data. The model was formalized by Henzinger, Raghavan, and Rajagopalan [9], who gave several algorithms and complexity results related to graph-theoretic problems and their applications. Other recent results on data streams can be found in [6, 18, 19, 5, 10]. The work of Feigenbaum et al [5, 10] constructs sketches of data-streams, under the assumption that the input is ordered by the adversary. Here we intend to succinctly capture the input data in histograms, thus the attribute values are assumed to be indexed in time. This is mostly motivated from concerns of modeling time series data, its representation, and storage.

. Most of these histogram problems can be solved offline using dynamic programming. The best known results require quadratic time and linear space to construct the optimal solution. We are assuming that the size of histogram is a small constant. We provide a $1+\epsilon$ approximation that runs in linear time. Moreover, our algorithm works in the data-stream model with space polylogarithmic in the number of items. The saving in space is a significant aspect of construction of histograms, since typically the number of items is large, and hence the requirement of approximate description. We then generalize the least square histogram construction to a broader class of partitioning in intervals. The only restriction we have is that the cost of approximating intervals is monotonic under inclusion, and can be computed from a small amount of information. This allows us to apply the result immediately to get approximations for splines, block compression amongst others. It is interesting that such a nicely described class can be approximated under data-streams algorithms.

Organization of the Paper: . In the next section we discuss the better known histogram optimality criteria, and in Section 3 we show how to reduce the construction time from quadratic to linear for the least squares error fit (V-histogram). We then generalize the result to a fairly generic error condition in Section 4.

2. PROBLEM STATEMENT

In this paper we will be considering serial histograms. Let us assume that the data is a sequence of non-negative integers $v_1, \ldots, v_n$. We are to partition the index set $1..n$ into k intervals or buckets minimizing $\sum_p \text{VAR}_p$ where VAR_p is the variance of the values in the p'th bucket. In a sense we are approximating a curve by a k-step function and the error is the least square fit. This is known as V-optimal histogram or V-histogram. We will concentrate on this problem for the most part. In the context of databases this arises due to the sum of the squares of errors in answering point queries on the database, the histogram provides an estimate for v_i. A similar question can be posed in terms of answering aggregate queries, trying to estimate $\sum_i v_i$.

Several other relevant measures of histogram exist, see [26]. Instead of minimizing the least square fit, it can be the area of the symmetric difference (sum of differences as opposed to their squares), related to the number of distinct values in a bucket (compressed histograms, biased or unbiased). The dual problem of minimizing the number of buckets, while constraining the error has been considered in [15, 23].

Previous Results. In [15] a straightforward dynamic programming algorithm was given to construct the optimal V-histogram. The algorithm runs in time $O(n^2k)$ and required $O(n)$ space. The central part of the dynamic programming approach is the following equation:

$$\text{OPT}[k,n] = \min_{x<n} \{\text{OPT}[k-1,x] + \text{VAR}[(x+1)..n]\} \qquad (1)$$

In the above equation, $\text{OPT}[p,q]$ represents the minimum cost of representing the set of values indexed by $1..q$ by a histogram with p levels. $\text{VAR}[a..b]$ indicates the variance of the set of values indexed by $a..b$.

The index x in the above equation can have at most n values, thus each entry can be computed in $O(n)$ time. With $O(nk)$ entries, the total time taken is $O(n^2k)$. The space required is $O(n)$; since at any step we would only require information about computing the variance and $\text{OPT}[p,x]$ and $\text{OPT}[p+1,x]$ for all x between 1 and n. The variance can be computed by keeping two $O(n)$ size arrays storing the prefixed sums $\sum_{i=1}^{x} v_i$ and $\sum_{i=1}^{x} v_i^2$.

3. FASTER CONSTRUCTION OF NEAR OPTIMAL V-HISTOGRAMS

In this section we will demonstrate a $1+\epsilon$ approximation of the optimal V-histogram, which will require linear time and only polylogarithmic space. Of course throughout the paper we will assume that k, the number of levels in the histogram, is small since that is the central point of representation by histograms. In the first step we will explain why we expect the improvement, and then proceed to put together all the details. We will also explain why the result immediately carries over to a data-stream model of computation.

Intuition of Improvement

Let us reconsider equation 1. The first observation we can make is:

$$\text{For } a \le x \le b\ , \quad \text{VAR}[a..n] \ge \text{VAR}[x..n] \ge \text{VAR}[b..n]$$

And since any solution for the index set $1..b$ also gives a feasible (may not be optimal) solution for the index set $1..x$

for any $x \leq b$, the second observation is

$$\text{For } a \leq x \leq b \ , \quad \text{Opt}[p, a] \leq \text{Opt}[p, x] \leq \text{Opt}[p, b]$$

This is a very nice situation, we are searching for the minimum of the sum of two functions, one of which is non-increasing (Var) and the other non-decreasing ($\text{Opt}[p, x]$ as x increases). A natural idea would be to use the monotonicity to reduce the searching to logarithmic terms. However this is not true, it is easy to see that to find the *true* minimum we have to spend $\Omega(n)$ time.

To see this, consider a set of non-negative values $v_1, \ldots, v_n$. Let $f(i) = \sum_{r=1}^{i} v_r$. Consider the two functions $f(i)$ and $g(i)$, where $g(i)$ is defined to be $f(n) - f(i-1)$. The function $f(i)$ and $g(i)$ are monotonically increasing and decreasing respectively. But to find the minimum of the sum of these two functions, amounts to minimizing $f(n) + v_i$, or in other words minimizing v_i.

However the redeeming aspect of the above example is the $f(n)$ part, picking any i gives us a 2 approximation. In essence this is what we intend to use, that the searching can be reduced if we are willing to settle for approximation. Of course we will not be interested in a 2 approximation but a factor of $1 + \epsilon$; the central idea would be the reduction in search. We will later see that this would generalize to optimizations over contiguous intervals of the indices.

Putting Details Together

We now provide the full details of the $1 + \epsilon$ approximation. We will introduce some small amount of notation. Our algorithm will construct a solution with p levels for every p and x with $1 \leq x \leq n$ and $1 \leq p \leq k$. Let $\text{Sol}[p, x]$ denote the cost of our solution with a p level V-histogram over the values indexed by $1..x$.

The algorithm will inspect the values indexed in increasing order. Let the current index being considered is n. Let v_i be the value of the i'th item (indexed by i). The parameter δ will be fixed later. The algorithm will maintain intervals, for every $1 \leq p \leq k$, $(a_1^p, b_1^p), \ldots, (a_l^p, b_l^p)$, such that

1. The intervals are disjoint and cover $1..n$. Therefore $a_1^p = 1$, and $b_l^p = n$. Moreover, $b_j^p + 1 = a_{j+1}^p$ for $j < l$. It is possible that $a_j^p = b_j^p$.
2. We maintain $\text{Sol}[p, b_j^p] \leq (1 + \delta)\text{Sol}[p, a_j^p]$.
3. Thus we will store $\text{Sol}[p, a_j^p]$ and $\text{Sol}[p, b_j^p]$ for all j and p. We will also store $\sum_{i=1}^{r} v_i$ and $\sum_{i=1}^{r} v_i^2$ for each $r \in \cup_{p,j}\{\{a_j^p\} \cup \{b_j^p\}\}$.
4. Clearly the number of intervals l would depend on p. We will use l instead of $l(p)$, to simplify the notation. The appropriate p will be clear from the context. It is important to observe that the value of l is not fixed and is as large as required by the above conditions.

The algorithm on seeing the $n+1$'st value v_{n+1}, will compute $\text{Sol}[p, n+1]$ for all p between 1 and k, and update the intervals $(a_1^p, b_1^p), \ldots, (a_l^p, b_l^p)$. We discuss the later first. In fact the algorithm has to update only the last interval (a_l^p, b_l^p), either setting $b_l^p = n + 1$, or creating a new interval $l + 1$, with $a_{l+1}^p = b_{l+1}^p = n + 1$, depending on $\text{Sol}[p, n + 1]$. This brings us to the computation of $\text{Sol}[p, n + 1]$.

For $p = 1$, the value of $\text{Sol}[p, n+1]$ is simply $\text{Var}[1..n+1]$ which can be computed from the prefixed sums $\sum_i v_i$ and $\sum_i v_i^2$. To compute $\text{Sol}[p + 1, n + 1]$, we will find the j such that

$$\text{Sol}[p, b_j^p] + \text{Var}[(b_j^p + 1)..(n + 1)]$$

is minimized. This minimum sum will be the value of $\text{Sol}[p+1, n + 1]$.

The algorithm is quite simple. We will now show how the optimum solution behaves. But first observe that we are not referring to a value which we have not stored, hence the algorithm applies to the data-stream model modulo the storage required. The following theorem will prove the approximation guarantee.

Theorem 3.1. *The above algorithm gives a $(1 + \delta)^p$ approximation to* $\text{Opt}[p + 1, n + 1]$.

Proof. The proof will be by double induction. We will solve by induction on n fixing p first. Subsequently assuming the theorem to be true for all $p \leq k - 1$, we will show it to hold for all n with $p = k$.

For $p = 1$, we are computing the variance, and we will have the exact answer. Thus the theorem is true for all n, for $p = 1$.

We assume that the theorem is true for all n for $p = k-1$. We will show the theorem to be true for all n for $p = k$.

Consider equation 1 (altering it to incorporate $n + 1$ instead of n) and let x be the value which minimizes it. In fact $(x+1)..(n+1)$ is the last interval in the optimum's solution to $\text{Opt}[k, n + 1]$. Consider $\text{Opt}[k - 1, x]$, and the interval (a_j^{k-1}, b_j^{k-1}) in which x lies. From the monotonicity of Opt it follows that:

$$\text{Opt}[k - 1, a_j^{k-1}] \leq \text{Opt}[k - 1, x] \leq \text{Opt}[k - 1, b_j^{k-1}]$$

And by induction hypothesis, we have:

$$\text{Sol}[k - 1, b_j^{k-1}] \leq (1 + \delta)\text{Sol}[k - 1, a_j^{k-1}] \leq$$
$$(1 + \delta)\left[(1 + \delta)^{k-2}\text{Opt}[k - 1, a_j^{k-1}]\right]$$

Now we also have, $\text{Var}[(b_j^{k-1} + 1)..(n + 1)] \leq \text{Var}[(x + 1)..(n + 1)]$. Therefore putting it all together,

$$\text{Opt}[k, n + 1] = \text{Opt}[k - 1, x] + \text{Var}[(x + 1)..(n + 1)]$$
$$\geq (1 + \delta)^{-k}\text{Sol}[k - 1, b_j^{k-1}] + \text{Var}[(b_j^{k-1} + 1)..(n + 1)]$$

Our solution will be at most $\text{Sol}[k-1, b_j^{k-1}] + \text{Var}[(b_j^{k-1} + 1)..(n + 1)]$. This proves the theorem. $\square$

We will set δ such that $(1 + \delta)^k \leq 1 + \epsilon$. Thus setting $\log(1 + \delta) = \epsilon/k$ is sufficient. The space required will be $O(k \log \text{Opt}[k, n + 1]/\log(1 + \delta))$. Observe that if the optimum is 0, it will correspond to k different sets of contiguous integers, which will correspond to the intervals (a_1^p, b_1^p). Thus we will require $\log \text{Opt}[p, n+1]/\log(1+\delta)+1$ intervals for this p. Thus the total space requirement is (k^2/ϵ) times $O(\log \text{Opt}[k, n+1])$. Now the optimum can be at most nR^2 where R is the maximum value, and $\log R$ will be the space required to store this number itself. Thus $\log \text{Opt}[k, n + 1]$ will correspond to storing $\log n + 2$ numbers, the algorithm takes an incremental $O((k^2/\epsilon) \log n)$ time per new value. We can claim the following theorem:

THEOREM 3.2. *We can provide a $(1+\epsilon)$ approximation for the optimum V-histogram with k levels, in $O((k^2/\epsilon)\log n)$ space and time $O((nk^2/\epsilon)\log n)$, in the data-stream model.*

4. LINEAR PARTITIONING PROBLEMS

Consider a class of problems which can expressed as follows, that given a set of values $v_1, \dots, v_n$, it requires to partition the set into k contiguous intervals. The objective is to minimize a function (for example, the sum, max, any ℓ_p norm amongst others) of a measure on the interval. In the case of V-histograms, the function is the sum and the measure on each interval is the variance of the values.

If the measure can be approximated upto a factor ρ for the interval, dynamic programming will give a ρ approximation for the problem. Moreover if this computation can be performed in constant time (for example the variance, from knowing $\sum_i v_i$ and $\sum_i v_i^2$) the dynamic programming will require only quadratic time. In case of histograms, the variance, was computed exactly and we had an optimal algorithm. We will claim a meta-theorem below, which generalizes the previous section, and see what results follow from it.

THEOREM 4.1. *The algorithm in the last section generalizes to give us a $(1+\epsilon)\rho$ algorithm in time $\tilde{O}(nk^2/\epsilon)$ for data-streams.*

We will use the theorem repeatedly in several situations, first we will consider a case where we cannot compute the error in a bucket exactly (in case of streams).

Approximating the error within a bucket

Consider the objective function where instead of minimizing the variance over an interval, we minimize $\sum_i |v_i - z|$ where z is the value stored in the histogram for this interval. In this case the value of z is the median of the (magnitude of) values. This corresponds to minimizing the area of the symmetric difference of the data and the approximation. This is an ℓ_1 norm instead of variance which is the square of ℓ_2. Using the results of [18], we can find an element of rank $n/2 \pm n\epsilon$ (assuming this interval has n elements) in space $O(\log n/\epsilon)$. Thus with the increase in space, we have an $(1+\epsilon)$ approximation. (Actually we need to choose space according to $\epsilon/3$, for the factor to work out.)

THEOREM 4.2. *In case of ℓ_1 error, we can approximate the histogram upto a factor of $1+\epsilon$ in space $O((k/\epsilon)^2 \log^2 n)$, and time $O(n(k/\epsilon)^2 \log^2 n)$ for data-streams.*

Approximating by piecewise splines

Another example of Theorem 4.1 will be to approximate by piecewise splines of small fixed degree instead of piecewise constant segments. The objective would be to minimize sum of squares fit for the points. We can easily convince ourselves that given a set of n values indexed by some integer range $a..b$, we can construct the coefficients of the best fit curve in time independent of n by maintaining prefix sums appropriately.

Consider the case of piecewise linear functions. Given a set of values v_i over a range $i \in a..b$, if the function is $s \cdot i + t$, it is not difficult to see that setting t to the mean and storing $\sum_i iv_i$ along with $\sum v_i$ and $\sum_i v_i^2$ we can minimize the expression by varying s. This generalizes to degree d splines; we claim the following without the calculations details,

THEOREM 4.3. *By storing $\sum_i i^d v_i, \dots, \sum_i v_i$ and $\sum_i v_i^2$ the above algorithm allows to compute a $1+\epsilon$ approximation in time $O((dnk^2/\epsilon)\log n)$ and space $O((dk^2/\epsilon)\log n)$ for data streams.*

Relaxing uniformity within buckets

Suppose we want to divide into partitions such that each part has approximately the same number of distinct values, and we want to minimize the maximum number of different values in a partition. This appears in cases where we are reluctant to make the uniformity assumption that any value in a bucket is equally likely. In such cases each bucket is tagged with a list of the values present, this helps in compression of the stream; this is typically the compressed histogram approach [26].

THEOREM 4.4. *We can approximate the above upto $(1+\epsilon)$ fraction if the (integer) values occur in a bounded range (say $1..R$), in space $O((Rk^2/\epsilon)\log n)$, for a data-stream.*

However in these approaches often biased histograms are preferred in which the highest occurring frequencies are stored (under belief of zipfian modeling, that they constitute most of the data) and the rest of the values are assumed uniform. Even this histogram can be constructed upto $1+\epsilon$ factor, under the assumption of a bounded range of values. Under the unbounded assumption, estimating the number of distinct values itself is quite hard [4].

Finding block boundaries in compression

In text compression (or compressing any discrete distribution) in which few distinct values appear in close vicinity, it is conceivable that partitioning the data in blocks, and compressing each partition is beneficial. The Theorem 4.1 allows us to identify the boundaries (upto $1+\epsilon$) loss in the optimal space requirement. This is because the space required by an interval obeys the measure property we require, namely that the cost of an interval is monotonic under inclusion.

Aggregate queries and histograms

The aggregate function can be treated as a curve, and we can have a $1+\epsilon$ approximation by a V-histogram with k levels. However there is a problem with this approach. If we consider the distribution implied by the data, it approximates the original data to k values nonzero values; since the aggregate will have only k steps. It appears more reasonable to approximate the aggregate by piecewise linear functions. In fact for this reason, most approaches prefer to approximate the data by a histogram, and use it for aggregate purposes. However it is not difficult to see that a V-histogram need not minimize the error. The V-histogram sums up the square errors of the point queries, our objective function would be to sum up square errors of aggregate queries. Thus if we approximate $v_q, \dots, v_r$ by z (in the data) and a shift t, the sum of square error for aggregate queries is

$$\sum_{q \le i \le r} \Big(\sum_{q \le j \le i} v_i - i \cdot z - t \Big)^2$$

Of course this assumes that any point is equally likely for the aggregate, and that we are answering aggregate from the first index. This is the same as in Section 3 except the incoming values can be thought of as $v_i' = \sum v_i$.

Thus we can have a $1+\epsilon$ approximation for this objective. In fact we can optimize a linear combination of the errors of point queries and the aggregate queries, where the z is provided as an answer to a point query and $zi + t$ as an answer to the aggregate. This may be useful in cases where the mix of the types of queries are known. These queries have been referred to as prefix queries in [17] in context of hierarchical range queries and were shown to computable in $O(n^2k)$ time.

Acknowledgements

We would like to thank S. Kannan, S. Muthukrishnan, V. Teague for many interesting discussions.

5. REFERENCES

[1] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. Join Synopses For Approximate Query Answering. *Proceedings of ACM SIGMOD*, pages 275–286, June 1999.

[2] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. *Proceedings of ACM SIGMOD*, pages 574–578, June 1999.

[3] M. Bertolotto and M. J. Egenhofer. Progressive vector transmission. *Proceedings of the 7th ACM symposium on Advances in Geographical Information Systems*, 1999.

[4] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough ? *Proceedings of ACM SIGMOD*, 1998.

[5] J. Feigenbaum, S. Kannan, M. Strauss, and M. Vishwanathan. An approximate l^1–difference algorithm for massive data sets. *Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 501–511, 1999.

[6] P. Flajolet and G. N. Martin. Probabilistic counting. *Proceedings of 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 76–82, 1983.

[7] P. Gibbons, Y. Mattias, and V. Poosala. Fast Incremental Maintenance of Approximate Histograms. *Proceedings of VLDB*, pages 466–475, 1997.

[8] J. Gray, A. Bosworth, A. Leyman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross Tab and Sub Total. *Proceedings of ICDE*, pages 152–159, 1996.

[9] M. R. Henzinger, P .Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-011, Digital Equipment Corporation, Systems Research Center*, May, 1998.

[10] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *Proceedings of 41st Annual Symposium on Foundations of Computer Science*, 2000.

[11] Y. Ioannidis. Universality of Serial Histograms. *Proceedings of VLDB*, pages 256–277, 1993.

[12] Y. Ioannidis and S. Christodoulakis. Optimal Histograms for Limiting Worst-Case Error Propagation in the Size of Join Results. *ACM Transactions on Database Systems, Vol. 18, No. 4*, pages 709–748, December 1993.

[13] Y. Ioannidis and V. Poosala. Histogram Based Approximation of Set Valued Query Answers. *Proceedings of VLDB*, pages 174–185, 1999.

[14] Y. Ioannidis and V. Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *Proceedings of ACM SIGMOD*, 1995.

[15] H. V Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal Histograms with Quality Guarantees. *Proceedings of VLDB*, 1998.

[16] H. V. Jagadish, Nick Koudas, and S. Muthukrishnan. Mining Deviants in a Time Series Database. *Proceedings of VLDB*, 1999.

[17] N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal histograms for hierarchical range queries. *Proceedings of ACM PODS*, 2000.

[18] G .S .Manku, S. Rajagopalan, and B. Lindsay. Approximate medians and other quantiles in one pass with limited memory. *Proceedings of the ACM SIGMOD*, 1998.

[19] G .S .Manku, S. Rajagopalan, and B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large databases. *Proceedings of ACM SIGMOD*, 1999.

[20] Y. Matias, J. Scott Vitter, and M. Wang. Wavelet-Based Histograms for Selectivity Estimation. *Proceedings of ACM SIGMOD*, 1998.

[21] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, pages 315–323, 1980.

[22] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multidimension al queries. *Proceedings of ACM SIGMOD*, 1998.

[23] S. Muthukrishnan, V. Poosala, and T. Suel. On Rectangular Partitioning In Two Dimensions: Algorithms, Complexity and Applications. *Proceedings of ICDT*, 1999.

[24] V. Poosala and V. Ganti. Fast Approximate Answers To Aggregate Queries On A Datacube. *SSDBM*, pages 24–33, 1999.

[25] V. Poosala and Y. Ioannidis. Estimation of Query-Result Distribution and Its Application In Parallel Join Load Balancing. *Proceedings of VLDB*, 1996.

[26] V. Poosala and Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. *Proceedings of VLDB*, 1997.

[27] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of ACM SIGMOD*, 1996.

[28] J. Vitter and M. Wang. Approximate computation of multidimensional aggregates on sparse data using wavelets. *Proceedings of ACM SIGMOD*, 1999.

[29] J.S. Vitter, M. Wang, and B. R. Iyer. Data Cube Approximation and Histograms via Wavelets. *Proceedings of the 1998 ACM CIKM Intern. Conf. on Information and Knowledge Management*, 1998.