

Wavelet-Based Histograms for Selectivity Estimation ^{*}

Yossi Matias
Department of Computer Science
Tel Aviv University
Tel Aviv 69978, Israel
matias+www@math.tau.ac.il

Jeffrey Scott Vitter
Purdue University
1390 Mathematical Sciences Building
West Lafayette, IN 47909, USA
jsv@purdue.edu

Min Wang[†]
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532, USA
min@us.ibm.com
Phone: (914) 784-6268
FAX: (914) 784-7455

Abstract

Query optimization is an integral part of relational database management systems. One important task in query optimization is selectivity estimation. Given a query P , we need to estimate the fraction of records in the database that satisfy P . Many commercial database systems maintain histograms to approximate the frequency distribution of values in the attributes of relations.

In this paper, we present a technique based upon a multiresolution wavelet decomposition for building histograms on the underlying data distributions. Histograms built on the cumulative data distributions give very good approximations with limited space usage. We give fast algorithms for constructing histograms and using them in an on-line fashion for selectivity estimation. Our histograms can also be used to provide quick approximate answers to OLAP queries when the exact answers are not required. Our method captures the joint distribution of multiple attributes effectively, especially when the attributes are correlated. Experiments confirm that our histograms offer substantial improvements in accuracy over random sampling and other previous approaches.

^{*}A preliminary version of this paper was published at SIGMOD 1998 [MVW98].

[†]Contact author.

1 Introduction

Several important components in a database management system (DBMS) require accurate estimation of the selectivity of a given query. For example, query optimizers use it to evaluate the costs of different query execution plans and choose the preferred one. In this section we briefly review previous work on selectivity estimation.

1.1 System R

IBM's System R is among the earliest RDBMSs, and its optimizer has set a tone for most modern optimizers [SAC⁺79]. The optimizers of several major database products still follow the general framework set by System R. That is, for any given query, the optimizer will do the following:

- Define a query plan space and an algorithm to search and enumerate all plans in the space.
- Define a cost model by which the cost of a query plan can be estimated.
- When a user issues a query, the optimizer will search through its query plan space and estimate the cost of each plan. Then the best (least-cost) plan is chosen and given to a *query execution engine* to run and the output is returned to the user.

However, the cost model of System R is not good enough because its most important component, selectivity estimation, is too simple. The selectivity estimation is done based on very simplified and unrealistic assumptions about the underlying data distribution. For example, the query optimizer assumes that each column of a table has uniform distribution and the columns are independent of each other. As a result, the selectivity estimation may be very inaccurate, and the optimizer could choose very inefficient query plans.

1.2 Bucket Histograms

The histogram is the most widely used form to store statistical information of the data distributions in a database. Many different histograms have been proposed in the literature and some have been deployed in commercial RDBMSs. However, almost all previous histograms have one thing in common: they all use buckets to partition the data, although in different ways.

A bucket-based histogram approximates the data in an attribute of a relation by grouping attribute values into *buckets* and estimating the attribute values and their frequencies based on the summary statistics maintained in each bucket. More specifically, a histogram of an attribute

is an approximation of the frequency distribution of the attribute values obtained as follows: The $(\text{attribute value}, \text{frequency})$ pairs of the distribution are partitioned into buckets; The frequency of each value in a bucket is approximated by the average of the frequencies of all values in the bucket; The set of values in a bucket is usually approximated in some compact fashion as well.

The simplest histogram is the so-called *equi-width histogram*. It partitions the domain of an attribute into consecutive equal-length subintervals. Each subinterval is characterized by just one parameter: the average frequency of the values in the subinterval.

The popular *equi-depth histogram* [PSC84] partitions the interval between the minimum and maximum attribute value of a column into consecutive subintervals so that the total frequency of the attribute values for each subinterval is the same.

It is interesting to note that these two simple histograms (or their variants) are still used extensively in today's major commercial RDBMSs like Oracle, Informix, Microsoft SQL Server, Sybase, and Ingres.

Poosala et al. [PIHS96, Poo97a] propose a taxonomy to capture all previously proposed histograms; new histogram types can be derived by combining effective aspects of different histogram methods. Among the histograms discussed in [PIHS96, Poo97a], the $\text{MaxDiff}(V, A)$ histogram gives the best overall performance. The $\text{MaxDiff}(V, A)$ histogram uses attribute value V as “sort parameter” and *area* A as “source parameter” to choose the bucket boundaries. In a $\text{MaxDiff}(V, A)$ histogram with β buckets, there is a bucket boundary between two adjacent attribute values if the difference between their areas is one of the $\beta - 1$ largest such differences. It is shown in [PIHS96, Poo97a] that $\text{MaxDiff}(V, A)$ histograms achieve better approximation than previous well-known methods like equi-depth and compressed histograms [PIHS96].

Poosala et al. [PIHS96] also mention other histogram techniques. For example, histograms based on minimizing the variance of the source parameter such as area have a performance similar to that of $\text{MaxDiff}(V, A)$, but are computationally more expensive to construct. They also mention a spline-based technique, but it does not perform as well as $\text{MaxDiff}(V, A)$.

All the histograms discussed so far partition the underlying data in some particular fashion. The effectiveness and accuracy of various partition methods highly depend on the data distributions and query types. In general, using a specific partition method to capture many unpredictable data distributions is a hard job, especially for multidimensional data. It is desirable to study general methods to construct histograms whose usability is more robust.

1.3 Non-Histogram Techniques

Random sampling has attracted extensive study in database community in the last 15 years and there has been considerable amount of work done in sampling-based estimations [HS92, HS95, LNS90, LN85, GM98]. All sampling-based techniques are based on the fact that a large data set can be represented well by a small random sample of the data elements. For selectivity estimation, sampling is a natural tool: at query optimization time, a random sample is collected and the selectivity of any given query is computed based on the sample. The advantages of sampling-based methods are obvious: no pre-computed and stored statistical information is needed, and it has probabilistic guarantees on the accuracy. However, obtaining a good sample at query optimization time is expensive, and sampling results cannot be preserved for later uses.

Another estimation technique is *probabilistic counting* and it is mainly used for estimating the size of projections. We refer the readers to [FM85, SDNR96, GG81] for more details.

Parametric estimation is a standard statistical technique that can also be used to estimate selectivity [Chr83, SLRD83]. However, this technique assumes that the data distribution resemble some well studied mathematical distribution that is usually characterized by several parameters. The assumption is often not true in real database applications.

1.4 Multidimensional Data

When a query involves multiple attributes in a relation, the selectivity depends on these attributes' *joint data distribution*, that is, the frequencies of all combination of attribute values. The main challenge for histograms for multidimensional (multi-attribute) data is to capture the correlations among different attributes.

To simplify the selectivity estimation for multi-attribute queries, most commercial DBMSs make the *attribute value independence assumption*. Under such an assumption, a system maintains histograms only for individual attributes, and the joint probabilities are derived by multiplying the individual probabilities. Real-life data rarely satisfy the attribute value independence assumption. Functional dependencies and various types of correlations among attributes are very common. Making the attribute value independence assumption in these cases results in very inaccurate estimation of the joint data distribution and poor selectivity estimation.

Muralikrishna and DeWitt [MD88] use an interesting spatial index partitioning technique to construct equi-depth histograms for multidimensional data. One drawback with this approach

is that it considers each dimension only once during the partition. Poosala and Ioannidis [PI97] propose two effective alternatives. The first approach partitions the joint data distribution into mutually disjointed buckets and approximates the frequency and the value sets in each bucket in a uniform manner. Among this new class of histograms, the multidimensional MaxDiff(V,A) histograms computed using the MHIST-2 algorithm are the most accurate and perform better in practice than previous methods [PI97]. The second approach uses the powerful *singular value decomposition* (SVD) technique from linear algebra. The approach is limited to handling two dimensions, and its accuracy depends largely on that of the underlying one-dimensional histograms.

1.5 Contributions

The contribution of this paper is threefold. First, we propose a method to build efficient and effective histograms using wavelet decomposition. Our histograms give accurate selectivity estimation for query optimization and other related applications in database systems. Second, our method can be naturally extended to capture the distribution of multidimensional data. Third, we introduce the structure of error tree to represent wavelet decomposition procedure. Based on this structure, we develop efficient reconstruction algorithm for query optimization time.

Subsequent to the introduction of wavelet-based selectivity estimation in the preliminary version of this paper [MVW98], there have been many interesting new developments in using wavelet-based techniques in databases and related applications [Wan99, VWI98, VW99, CGRS00, MVW00, GG02, NRS99, GKMS01, WVLP, DR, PM, MP03, BMW]; further elaboration is given in Section 6. Many of these developments are based on the basic techniques introduced in this paper.

The rest of the paper is organized as follows. In Section 2, we give a brief introduction on wavelets and explain in detail how to build a wavelet-based histogram. In Section 3, we explain how to estimate the selectivity of any given range query using our wavelet-based histogram at query optimization time. In Section 4, we extend our work to multi-dimensional cases. We compare the performance of our wavelet-based histogram with random sampling and state-of-the-art histogram method through experiments in Section 5 and draw conclusions in Section 6.

2 Wavelet-Based Histograms

In this section we describe how to construct a wavelet-based histogram. We first give a high-level description of the construction algorithm. We then describe the algorithm steps in detail.

Wavelets are a mathematical tool for hierarchical decomposition of functions. Wavelets represent a function in terms of a coarse overall shape, plus details that range from broad to narrow. Regardless of whether the function of interest is an image, a curve, or a surface, wavelets offer an elegant technique for representing the various levels of detail of the function in a space-efficient manner.

A histogram can be considered as a compressed representation of the original data. In this sense, our approaches can also be categorized as histogram methods. However, our methods neither use buckets nor do they try to partition the original data. Instead, our methods try to extract and summarize information from the original data. Although the traditional histograms also summarize the original data, they are *local*, while our methods are *hierarchical and global*.

2.1 Notations

The set of predicates we consider in this paper for purpose of estimating data distribution is the set of selection queries, in particular, *range predicates* of the form $l \leq X \leq h$, where X is a non-negative attribute of the domain of a relation R , and l and h are constants. The predicate is satisfied by those tuples for which the value of attribute X is between l and h . The set of *equal predicates* is the subset of the range predicates that have $l = h$. The set of *one-side range predicates* is the special case of range predicates in which $l = -\infty$ or $h = \infty$.

We adopt the notations in [PIHS96] to describe the data distributions and various histograms. The *domain* $D = \{0, 1, 2, \dots, N - 1\}$ of an attribute X is the set of all possible values of X . The value set $V \subseteq D$ consists of the n distinct values of X that are actually present in relation R . Let $v_1 < v_2 < \dots < v_n$ be the n values of V . The *spread* s_i of v_i is defined as $s_i = v_{i+1} - v_i$. (We take $s_0 = v_1$ and $s_n = 1$.) The *frequency* f_i of v_i is the number of tuples in which X has value v_i . The *cumulative frequency* c_i of v_i is the number of tuples $t \in R$ with $t.X \leq v_i$; that is, $c_i = \sum_{j=1}^i f_j$. The data distribution of X is the set of pairs $\mathcal{T} = \{(v_1, f_1), (v_2, f_2), \dots, (v_n, f_n)\}$. The *cumulative data distribution* of X is the set of pairs $\mathcal{T}^C = \{(v_1, c_1), (v_2, c_2), \dots, (v_n, c_n)\}$. The *extended cumulative data distribution* of X , denoted by \mathcal{T}^{C+} , is the cumulative data distribution of \mathcal{T}^C extended over the entire domain D by assigning zero frequency to every value in $D - V$.

2.2 Building A Histogram: A High Level Description

Building a wavelet-based histogram is a three-step procedure:

1. *Preprocessing* In a preprocessing step, we form the *extended cumulative data distribution* \mathcal{T}^{c^+} of the attribute X , from the original data or from a random sample of the original data.
2. *Wavelet Decomposition* We compute the wavelet decomposition of \mathcal{T}^{c^+} , obtaining a set of N wavelet coefficients.
3. *Pruning* We keep only the m most significant wavelet coefficients, for some m that corresponds to the desired storage usage. The choice of which m coefficients we keep depends upon the particular pruning method we use.

After applying the above algorithm, we obtain m wavelet coefficients. The values of these coefficients, together with their positions (indices), are stored and serve as a histogram for reconstructing the approximate data distribution in the on-line phase (query phase). To compute the estimate for the number of tuples whose X value is in the range $l \leq X \leq h$, we reconstruct the approximate values for h and $l - 1$ in the extended cumulative distribution function and then subtract them.

Next we discuss each step in greater details.

2.3 Preprocessing

In the preprocessing step, we compute the full data distribution \mathcal{T} . The extended cumulative data distribution \mathcal{T}^{c^+} can be easily computed from \mathcal{T} . Exact computation of \mathcal{T} can be done by maintaining a counter for each distinct attribute value in V . When the cardinality of V is relatively small so that \mathcal{T} fits in internal memory, which is often the case for the low-dimensional data we consider, we can keep a (hash) table in memory, and \mathcal{T} can be obtained in one complete scan through the relation.

When the cardinality of V is too large, multiple passes through the relation will be required to obtain \mathcal{T} , resulting in possibly excessive I/O cost. We can instead use an I/O-efficient external merge sort to compute \mathcal{T} and minimize the I/O cost [Vit01]. The merge sort process here is different from the traditional one: During the merging process, records with the same attribute

value can be combined by summing the frequencies. After a small number of multi-way passes in the merge sort, the lengths of the runs will stop increasing; the length of each run is bounded by the cardinality of V , whose size, although too large to fit in memory, is typically small in comparison with the relation size.

If for any reason it's necessary to further reduce the I/O and CPU costs of the precomputation of \mathcal{T} , a well-known approach is to use random sampling [PSC84, MD88]. The idea is to sample s tuples from the relation randomly and compute \mathcal{T} for the sample. The sample data distribution is then used as an estimate of the real data distribution. To obtain the random sample in a single linear pass, the method of choice is the skip-based method [Vit87] when the number of tuples T is known beforehand or the reservoir sampling variant [Vit85] when T is unknown. A running up-to-date sample can be kept using a backing sample approach [GMP97]. We do not consider in this paper the issues dealing with sample size and the errors caused by sampling. Our experiments confirm that wavelet-based histograms that use random sampling as a preprocessing step give estimates that are almost as good as those from wavelet-based histograms that are built on the full data. On the other hand, as we shall see in Section 5, the wavelet-based histograms (whether they use random sampling in their preprocessing or not) perform significantly better at estimation than do naive techniques based on random sampling alone.

In this paper, we do not consider the I/O complexities of the wavelet decomposition and of pruning, since both of them are performed on the cumulative data distribution \mathcal{T}^{c+} , and the size of \mathcal{T}^{c+} is generally not large and can fit in internal memory for the low-dimensional data we considered in this paper. For high-dimensional data, which is generally larger, the I/O efficiencies of the wavelet decomposition and of pruning are considered in [VWI98, VW99].

2.4 Wavelet Decomposition

After the preprocessing step, we obtain the extended cumulative data distribution \mathcal{T}^{c+} . We can view \mathcal{T}^{c+} as a one-dimensional array S of size N . The goal of the wavelet decomposition step is to represent the array S at hierarchical levels of detail.

First we need to choose wavelet basis functions. Haar wavelets are conceptually the simplest wavelet basis functions, and for purposes of exposition in this paper, we focus our discussion on Haar wavelets. Suppose S is an array of N values:

$$S = [S(0), S(1), \dots, S(N - 1)].$$

To apply the Haar wavelet decomposition to S , we assume N is a power of 2. (Otherwise we can pad S with dummy zero entries. There are better methods than padding to use in practice.) We first normalize S at base level $j = \log N$ by setting $S_{j,k} = \frac{S(k)}{\sqrt{2^j}}$ for each $0 \leq k < N$.

The decomposition of array S can be done using the following recursive formulas:

$$\begin{aligned} S_{i,k} &= \frac{\sqrt{2}(S_{i+1,2k} + S_{i+1,2k+1})}{2} \\ \widehat{S}_{i,k} &= \frac{\sqrt{2}(S_{i+1,2k} - S_{i+1,2k+1})}{2} \end{aligned}$$

for $0 \leq i \leq j - 1$ and $0 \leq k \leq 2^i - 1$. The decomposition results is the sequence

$$[S_{0,0}, \widehat{S}_{1,0}, \widehat{S}_{1,1}, \dots, \widehat{S}_{j-1,2^{j-1}-1}].$$

The entries in the sequence are called *wavelet coefficients*.

The reconstruction is a reverse process to the decomposition, and can be done using the following backward recursive formulas:

$$\begin{aligned} S_{i,2k} &= \frac{\sqrt{2}(S_{i-1,k} + \widehat{S}_{i-1,k})}{2}; \\ S_{i,2k+1} &= \frac{\sqrt{2}(S_{i-1,k} - \widehat{S}_{i-1,k})}{2}, \end{aligned}$$

for $1 \leq i \leq j$ and $0 \leq k \leq 2^{i-1} - 1$.

The following pseudocode accomplishes the decomposition *in situ*, transforming a set of values into wavelet coefficients in place.

Procedure *Decomposition*(s : **array**[$0 \dots 2^j - 1$] **of reals**)

$s = s/\sqrt{2^j}$; //normalize input values

$g = 2^j$;

while $g \geq 2$ **do**

$DecompositionStep(s[0 \dots g - 1])$;

$g = g/2$;

Procedure *DecompositionStep*(s : **array**[$0 \dots 2^j - 1$] **of reals**)

for $i = 0$ **to** $2^j/2 - 1$ **do**

$s'[i] = s[2i] + s[2i + 1]/\sqrt{2}$;

$s'[i + 2^j/2] = s[2i] - s[2i + 1]/\sqrt{2}$;

$s = s'$;

The following pseudocode reconstructs the original data from the wavelet coefficients:

Procedure *Reconstruction*(s : **array**[$0 \dots 2^j - 1$] **of reals**)

```

 $s = s/\sqrt{2^j}$ ;
 $g = 2$ ;
while  $g \leq 2^j$  do
    ReconstructionStep( $s[0 \dots g - 1]$ );
     $g = 2g$ ;
 $s = s'\sqrt{2^j}$ ;           //undo normalization

```

Procedure *ReconstructionStep*(s : **array**[$0 \dots 2^j - 1$] **of reals**)

```

for  $i = 0$  to  $2^j/2 - 1$  do
     $s'[2i] = s[i] + s[2^j/2 + i]/\sqrt{2}$ ;
     $s'[2i + 1] = s[i] - s[2^j/2 + i]/\sqrt{2}$ ;
 $s = s'$ ;

```

The wavelet decomposition (reconstruction) is very efficient computationally, requiring only $O(N)$ CPU time for an array of N values.

The above decomposition and reconstruction processes include the normalization operation. The input array has been normalized in the very beginning. In the following, we illustrate the process of Haar wavelet decomposition using a concrete example. To simplify the presentation, we ignore the normalization factors.

Example 1 Suppose the extended cumulative data distribution of attribute X is represented by a one-dimensional array of $N = 8$ data items:

$$S = [2, 2, 2, 3, 5, 5, 6, 6].^1$$

We perform a wavelet transform on it. We first average the array values, pairwise, to get the new lower-resolution version with values

$$[2, 2.5, 5, 6].$$

That is, the first two values in the original array (2 and 2) average to 2, and the second two values 2 and 3 average to 2.5, and so on. Clearly, some information is lost in this averaging process. To recover the original array from the four averaged values, we need to store some *detail coefficients*, which capture the missing information. Haar wavelets store one half of the pairwise differences

¹The domain of attribute X is $D = \{0, 1, 2, \dots, 7\}$. The corresponding data distribution and extended cumulative data distribution of X are $\{(0, 2), (3, 1), (4, 2), (6, 1)\}$ and $\{(0, 2), (1, 2), (2, 2), (3, 3), (4, 5), (5, 5), (6, 6), (7, 6)\}$, respectively.

of the original values as detail coefficients. In the above example, the four detail coefficients are $(2 - 2)/2 = 0$, $(2 - 3)/2 = -0.5$, $(5 - 5)/2 = 0$, and $(6 - 6)/2 = 0$. It is easy to see that the original values can be recovered from the averages and differences.

We have succeeded in decomposing the original array into a lower-resolution version of half the number of entries and a corresponding set of detail coefficients. By repeating this process recursively on the averages, we get the full decomposition:

Resolution	Averages	Detail Coefficients
8	[2, 2, 2, 3, 5, 5, 6, 6]	
4	[2, $2\frac{1}{2}$, 5, 6]	[0, $-\frac{1}{2}$, 0, 0]
2	[$2\frac{1}{4}$, $5\frac{1}{2}$]	[$-\frac{1}{4}$, $-\frac{1}{2}$]
1	[$3\frac{7}{8}$]	[$-3\frac{1}{4}$]

We have obtained the *wavelet transform* (also called *wavelet decomposition*) of the original eight-value array, which consists of the single coefficient representing the overall average of the original values, followed by the detail coefficients in the order of increasing resolution:

$$\hat{S} = [3\frac{7}{8}, -3\frac{1}{4}, -\frac{1}{4}, -\frac{1}{2}, 0, -\frac{1}{2}, 0, 0]. \quad (1)$$

No information has been gained or lost by this process. The original array has eight values, and so does the transform. Given the transform, we can reconstruct the exact values by recursively adding and subtracting the detail coefficients from the next-lower resolution. \square

As illustrated in the pseudocode (but not the example), the wavelet coefficients at each level of the recursion are normalized; the coefficients at the lower resolutions are weighted more heavily than the coefficients at the higher resolutions. One advantage of the normalized wavelet transform is that in many cases a large number of the detail coefficients turn out to be very small in magnitude. Truncating these small coefficients from the representation (i.e., replacing each one by 0) introduces only small errors in the reconstructed signal. We can approximate the original signal effectively by keeping only the most significant coefficients, determined by some pruning method, as discussed in the next subsection.

A better higher-order approximation for purposes of range query selectivity can be obtained by using linear wavelets as a basis rather than Haar wavelets. Linear wavelets share the important properties of Haar wavelets that we exploit for efficient processing. It is natural in conventional histograms to interpolate the values of items within a bucket in a uniform manner. Such an approximation corresponds to a linear function between the endpoints of the bucket. The approximation induced when we use linear wavelets is a piecewise linear function, which implies

exactly this sort of linear interpolation. It therefore makes sense intuitively that the use of linear wavelets, in which we optimize directly for the best set of interpolating segments, will perform better than standard histogram techniques.

2.5 Pruning and Error Measures

Given the storage limitation for the histogram, we can only “keep” a certain number of the N wavelet coefficients. Let m denote the number of wavelet coefficients that we have room to keep; the remaining wavelet coefficients will be implicitly set to 0. Typically we have $m \ll N$. The goal of pruning is to determine which are the “best” m coefficients to keep, so as to minimize the error of approximation.

The problem is how to keep the “best” m coefficients by pruning out $N - m$ “insignificant” coefficients. To this end, we need to define what “best” means and what error measure we are going to use so that the pruning will minimize the error of approximation.

We can measure the error of approximation in several ways. Let v_i be the actual size of a query q_i and let \hat{v}_i be the estimated size of the query. We use the following five different error measures:

	Notation	Definition
<i>absolute error</i>	e_i^{abs}	$ v_i - \hat{v}_i $
<i>relative error</i>	e_i^{rel}	$\frac{ v_i - \hat{v}_i }{\max\{1, v_i\}}$
<i>modified relative error</i>	$e_i^{\text{m-rel}}$	$\frac{ v_i - \hat{v}_i }{\max\{1, \min\{v_i, \hat{v}_i\}\}}$
<i>combined error</i>	e_i^{comb}	$\min\{\alpha \times e_i^{\text{abs}}, \beta \times e_i^{\text{rel}}\}$
<i>modified combined error</i>	$e_i^{\text{m-comb}}$	$\min\{\alpha \times e_i^{\text{abs}}, \beta \times e_i^{\text{m-rel}}\}$

The parameters α and β are positive constants.

Our definition of relative error is slightly different from the traditional one, which is not defined when $v_i = 0$. The modified relative error treats over-approximation and under-approximation in a uniform way. For example, suppose the exact size of a query is $v_i = 10$. The estimated sizes $\hat{v}_i = 5$ or $\hat{v}_i = 20$ each have the same modified relative error, namely, $5/5 = 10/10 = 1$. In contrast, in terms of relative error, the estimation $\hat{v}_i = 5$ has a relative error of $5/10 = 0.5$, and the estimation $\hat{v}_i = 20$ has a relative error of $10/10 = 1$. The estimation $\hat{v}_i = 0$ has a relative error of only $10/10 = 1$, while the modified relative error is $10/1 = 10$. The combined error reflects the

importance of having either a good relative error or a good absolute error for each approximation. For example, for very small v_i it may be good enough if the absolute error is small even if the relative error is large, and for large v_i the absolute error may not be as meaningful as the relative error.

Once we choose which of the above measures to represent the errors of the individual queries, we need to choose a norm by which to measure the error of a collection of queries. Let $e = (e_1, e_2, \dots, e_Q)$ be the vector of errors over a sequence of Q queries. We assume that one of the above five error measures is used for each of the individual query errors e_i . For example, for absolute error, we can write $e_i = e_i^{\text{abs}}$. We define the overall error for the Q queries by one of the following error measures:

	Notation	Definition
<i>1-norm average error</i>	$\ e\ _1$	$\frac{1}{Q} \sum_{1 \leq i \leq Q} e_i$
<i>2-norm average error</i>	$\ e\ _2$	$\sqrt{\frac{1}{Q} \sum_{1 \leq i \leq Q} e_i^2}$
<i>infinity-norm error</i>	$\ e\ _\infty$	$\max_{1 \leq i \leq Q} \{e_i\}$

These error measures are special cases of the p -norm average error, for $p > 0$:

$$\|e\|_p = \left(\frac{1}{Q} \sum_{1 \leq i \leq Q} e_i^p \right)^{1/p}.$$

Given any particular weighting, we propose the following different pruning methods:

1. Choose the m largest (in absolute value) wavelet coefficients.
2. Choose m wavelet coefficients in a greedy way. For example, we might choose the m largest (in absolute value) wavelet coefficients and then repeatedly do the following two steps m times:
 - (a) Choose the wavelet coefficient whose inclusion leads to the largest reduction in error.
 - (b) Throw away the wavelet coefficient whose deletion leads to the smallest increase in error.

Another approach is to do the above two steps repeatedly until a cycle is reached or improvement is small.

Several other variants of the greedy method are possible:

3. Start with the $m/2$ largest (in absolute value) wavelet coefficients and choose the next $m/2$ coefficients greedily.
4. Start with the $2m$ largest (in absolute value) wavelet coefficients and throw away m of them greedily.

It is well known that if the wavelet basis functions are orthonormal then Method 1 is provably optimal for the 2-norm average error measure. However, for non-orthogonal wavelets like linear wavelets and for norms other than the 2-norm, no efficient technique is known for how to choose the m best wavelet coefficients, and various approximations have been studied [Don92].

At the end of the pruning step, we obtain m significant wavelet coefficients. These coefficients, together with their indices, form our wavelet-based histogram. Let us denote by H the histogram computed from a one-dimensional array S of length N . We can view H as a one-dimensional array of length m , with each entry being a wavelet coefficient value v_j and its index i_j

$$H[j] = (i_j, v_j), \quad 1 \leq j \leq m. \quad (2)$$

3 Online Reconstruction

At query optimization time, a range query $l \leq X \leq h$ is presented. We reconstruct the approximate cumulative frequencies of $l - 1$ and h , denoted by $S(l - 1)$ and $S(h)$, using the m wavelet coefficients we obtained in Section 2.5. The size of the query is estimated to be $S(h) - S(l - 1)$. The time for reconstruction is crucial in the on-line phase. In this section, we show how to efficiently compute the approximate cumulative frequency $S(i)$ (for any $0 \leq i \leq N - 1$).

To fully understand the on-line reconstruction algorithm, we examine the relationship between wavelet coefficients and the reconstruction of an original data value by using the *error tree* structure. The error tree is built based upon the wavelet transform procedure. Figure 1 is the error tree for Example 1 in Section 2.4. Each internal node is associated with a wavelet coefficient value, and each leaf is associated with an original signal value. (For purposes of exposition, the wavelet coefficients are unnormalized, but in the implementation the values are normalized as described in Section 2.4 and the algorithm is modified appropriately.) Internal nodes and leaves are labeled separately. Their labels are in the domain $\{0, 1, \dots, N - 1\}$ for a signal of length N . For example, the root is an internal node with label 0 and its node value is 3.875 in Figure 1. For convenience, we shall use “node” and “node value” interchangeably.

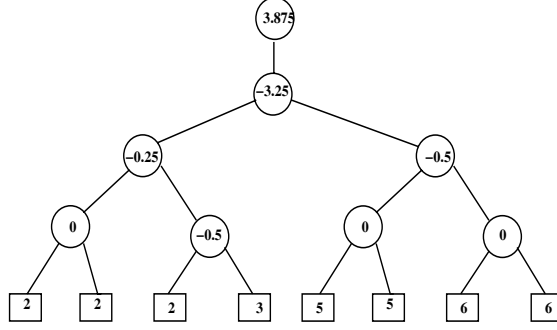


Figure 1: Error tree for $N = 8$.

The construction of the error tree exactly mirrors the wavelet transform procedure. It is a bottom-up process. First, leaves are assigned original signal values from left to right. Then wavelet coefficients are computed, level by level, and assigned to internal nodes.

As the figure shows, the (unnormalized) value of each internal node i is denoted by $\widehat{S}(i)$, and the value of each leaf j is denoted by $S(j)$. We use $left(i)$ and $right(i)$ to denote the left and right child of any node i , and we use $leaves(i)$ to denote the set of leaves in the subtree rooted at i . The average value of the nodes in $leaves(i)$ is denoted by $ave_leaf_val(i)$. For any leaf i , we use $path(i)$ to denote the set of internal nodes (or the node values) along the path from i to the root.

Below are some useful facts that are helpful for understanding our algorithm.

Lemma 1 *For any nonroot internal node i , we have*

$$\widehat{S}(i) = \frac{ave_leaf_val(left(i)) - ave_leaf_val(right(i))}{2}.$$

Lemma 2 *The reconstruction of any signal value depends only upon the values of those internal nodes along the path from the corresponding leaf to the root. That is, the reconstruction of any leaf value $S(i)$ depends only upon the nodes in $path(i)$.*

Consider the reconstruction of any leaf value $S(i)$. It is an algebraic sum of all the internal nodes in $path(i)$. For example, for $i = 1$,

$$S(1) = \widehat{S}(0) + \widehat{S}(1) + \widehat{S}(2) - \widehat{S}(4).$$

In general, any original signal value $S(i)$ can be represented as the algebraic sum of the wavelet coefficients along the path $path(i)$. A nonroot internal node contributes positively to the leaves

in its left subtree and negatively to the leaves in its right subtree. (The root node contributes positively to all the leaves.)

Lemma 3 *Mathematically, we can write any original signal value $S(i)$ in terms of all the wavelet coefficients in $path(i)$:*

$$S(i) = \sum_{j \in path(i)} sign(i, j) \times \widehat{S}(j),$$

where

$$sign(i, j) == \begin{cases} 1 & \text{if } j = 0 \text{ or leaf } i \text{ is in node } j \text{'s left subtree;} \\ -1 & \text{if leaf } i \text{ is in node } j \text{'s right subtree.} \end{cases}$$

The summation is over all internal nodes x in $path(i)$. In our algorithm, however, we do not evaluate all the terms. We quickly determine the nonzero contributors² and evaluate their contribution.

Let us relook at Example 1 in Section 2.4; its error tree is shown in Figure 1. The original signal S can be reconstructed from \widehat{S} by the following formulas:

$$\begin{aligned} S(0) &= \widehat{S}(0) + \widehat{S}(1) + \widehat{S}(2) && + \widehat{S}(4) \\ S(1) &= \widehat{S}(0) + \widehat{S}(1) + \widehat{S}(2) && - \widehat{S}(4) \\ S(2) &= \widehat{S}(0) + \widehat{S}(1) - \widehat{S}(2) && + \widehat{S}(5) \\ S(3) &= \widehat{S}(0) + \widehat{S}(1) - \widehat{S}(2) && - \widehat{S}(5) \\ S(4) &= \widehat{S}(0) - \widehat{S}(1) && + \widehat{S}(3) && + \widehat{S}(6) \\ S(5) &= \widehat{S}(0) - \widehat{S}(1) && + \widehat{S}(3) && - \widehat{S}(6) \\ S(6) &= \widehat{S}(0) - \widehat{S}(1) && - \widehat{S}(3) && + \widehat{S}(7) \\ S(7) &= \widehat{S}(0) - \widehat{S}(1) && - \widehat{S}(3) && - \widehat{S}(7) \end{aligned}$$

For example, $S(2)$ depends only upon $path(2) = \{\widehat{S}(5), \widehat{S}(2), \widehat{S}(1), \widehat{S}(0)\}$.

To estimate the selectivity of a range query of form $l \leq X \leq h$ using the m coefficients of our wavelet-based histogram H , we use the following algorithm:

Compute_Selectivity(H, m, l, h)

²The relevant internal nodes that are kept as significant coefficients after pruning step.


```

//compute the selectivity of range query  $l \leq x \leq h$ 
//using our wavelet-based histogram  $H$  that contains  $m$  coefficients
//initialize the approximate cumulative frequencies  $S(l-1)$  and  $S(h)$ 
 $S(l-1) = 0;$ 
 $S(h) = 0;$ 
for  $j = 1, 2, \dots, m$  do
  if  $Contribute(i_j, l-1)$  then
     $S(l-1) = S(l-1) + Compute\_Contribution(i_j, v_j, l-1)$ 
  if  $Contribute(i_j, h)$  then
     $S(h) = S(h) + Compute\_Contribution(i_j, v_j, h)$ 
selectivity =  $S(h) - S(l-1);$ 
return selectivity;

```

Function $Contribute(i, j)$ returns *true* if the coefficient with index i (internal node i in the error tree) contributes to the reconstruction of leaf value $S(j)$, and it returns *false* otherwise. Function $Compute_Contribution(i, v, j)$ computes the actual contribution of a coefficient (i, v) to the reconstruction of leaf value $S(j)$.

Based upon the regular structure of the error tree and the preceding lemmas, we devise two algorithms to compute the functions $Contribute$ and $Compute_Contribution$.

Function $Contribute(i, j)$

```

//check the coefficient with index  $i$  to see if it contributes
//to the reconstruction of leaf value  $S[j]$ 
//get the the label of the left most leaf for the subtree
//rooted at node  $i$  in an error tree with  $N$  leaves
 $L_l = left\_most\_leaf(i, N);$ 
//get the the label of the right most leaf for the subtree
//rooted at node  $i$  in an error tree with  $N$  leaves
 $L_r = right\_most\_leaf(i, N);$ 
if ( $j \in [L_l, L_r]$ ) //leave  $j$  is in the subtree
  then  $contribute = \mathbf{true}$ 
else //leave  $j$  is not in the subtree
   $contribute = \mathbf{false};$ 
return  $contribute;$ 

```

Function $Compute_Contribution(i, v, j)$

```

//compute the contribution of coefficient  $(i, v)$  to the reconstruction of leaf value  $S[j]$ 
  contribution = v;
  if (i = 0) //the special case for root node
    then return contribution;
  depth =  $\lfloor \log i \rfloor$ ; //compute the depth of node  $i$  in the error tree
  if (depth <  $\log N - 1$ ) //node  $i$  is not a parent of leaves
    then
      //get the the label of the left most leaf for node  $i$ 's right
      //subtree in an error tree with  $N$  leaves
       $L_{rl} = \text{left\_most\_leaf}(\text{right\_child}(i), N)$ ;
      //get the the label of the right most leaf for node  $i$ 's right
      //subtree in an error tree with  $N$  leaves
       $L_{rr} = \text{right\_most\_leaf}(\text{right\_child}(i), N)$ ;
    else //node  $i$  is a parent of leaves
       $L_{rl} = \text{right\_most\_leaf}(i, N)$ ;
       $L_{rr} = L_{rl}$ ;
  //use Lemma 3
  if  $j \in [L_{rl}, L_{rr}]$ 
    contribution = contribution  $\times (-1)$ ;
  return contribution;

```

Function $\text{left_most_leaf}(i, N)$

```

//compute the label of the left most leaf for subtree of node  $i$ 
//in an error tree with  $N$  leaves
//We denote the binary representation of  $i$  by  $(i)_2 = (b_{n-1} \dots b_2 b_1 b_0)$ ,
//where  $n = \log N$ ,  $b_j \in \{0, 1\}$  for  $0 \leq j \leq n - 1$ .
  if (i = 0) or (i = 1) //special cases
    then
      L = 0;
    return L;
  //p is the position of the left most 1 bit of  $(i)_2$ 
   $p = \max\{k \mid b_k = 1, 0 \leq k \leq n - 1\}$ ;
   $L = b_{p-1} \ll (n - p)$ ; // $\ll$  is the bitwise left shift operator
  return L;

```

Function *right_most_leaf*(i, N)
//compute the label of the right most leaf for subtree of node i
//in an error tree with N leaves
//We denote the binary representation of i by $(i)_2 = (b_{n-1} \dots b_2 b_1 b_0)$,
//where $n = \log N$, $b_j \in \{0, 1\}$ for $0 \leq j \leq n - 1$.
if $(i = 0)$ or $(i = 1)$ //special cases
then
 $R = N - 1$;
 return R ;
// p is the position of the left most 1 bit of $(i)_2$
 $p = \max\{k \mid b_k = 1, 0 \leq k \leq n - 1\}$;
 $L = b_{p-1} \ll (n - p)$; // \ll is the bitwise left shift operator
 $R = L + 2^{n-p} - 1$;
return R ;

Function *left_child*(i)
//compute the label (index) of the left child of a nonroot node i in an error tree
return $2i$;

Function *right_child*(i)
//compute the label (index) of the right child of a nonroot node i in an error tree
return $2i + 1$;

Theorem 1 *For a given range query of form*

$$l \leq X \leq h,$$

the approximate selectivity can be computed based upon the m coefficients of our wavelet-based histogram in $O(\min\{m, 2 \log N\})$ using a $2m$ -space data structure.

Proof Sketch: We need to process the m coefficients in H . Each of the coefficients is a tuple of the form (2), so the space complexity is $2m$. The CPU time complexity follows easily from that of the two functions, *Contribute* and *Compute_Contribution*, since both of them run in constant time. An alternative way is to process only the coefficients needed, which are at most $\log N$ for reconstructing either $S(l - 1)$ or $S(h)$. □

4 Multi-Attribute Histograms

In this section, we extend the techniques in previous section to deal with the multidimensional case.

We extend the definitions in Section 1 to the multidimensional case in which there are multiple attributes. Suppose the number of dimensions is d and the attribute set is $\{X_1, X_2, \dots, X_d\}$. Let $D_k = \{0, 1, \dots, N_k - 1\}$ be the domain of attribute X_k . The value set V_k of attribute X_k is the set of n_k values of X_k that are present in relation R . Let $v_{k,1} < v_{k,2} < \dots < v_{k,n_k}$ be the individual n_k values of V_k . The data distribution of X_k is the set of pairs $\mathcal{T}_k = \{(v_{k,1}, f_{k,1}), (v_{k,2}, f_{k,2}), \dots, (v_{k,n_k}, f_{k,n_k})\}$. The *joint frequency* $f(i_1, \dots, i_d)$ of the value combination $(v_{1,i_1}, \dots, v_{d,i_d})$ is the number of tuples in R that contain $v_{i_k,k}$ in attribute X_k , for all $1 \leq k \leq d$. The *joint data distribution* $\mathcal{T}_{1,\dots,d}$ is the entire set of (*value combination, joint frequency*) pairs. The joint frequency matrix $\mathcal{F}_{1,\dots,d}$ is a $n_1 \times \dots \times n_d$ matrix whose $[i_1, \dots, i_d]$ entry is $f(i_1, \dots, i_d)$. We can define the *cumulative joint distribution* $\mathcal{T}_{1,\dots,d}^C$ and *extended cumulative joint distribution* $\mathcal{T}_{1,\dots,d}^{C+}$ by analogy with the one-dimensional case. The extended cumulative joint frequency $\mathcal{F}_{1,\dots,d}^{C+}$ for the d attributes X_1, X_2, \dots, X_d is a $N_1 \times N_2 \times \dots \times N_d$ matrix S defined by

$$S[x_1, x_2, \dots, x_d] = \sum_{i_1=0}^{x_1} \sum_{i_2=0}^{x_2} \dots \sum_{i_d=0}^{x_d} f(i_1, i_2, \dots, i_d).$$

A very nice feature of our wavelet-based histograms is that they extend naturally to multiple attributes by means of multidimensional wavelet decomposition and reconstruction. The procedure of building the multidimensional wavelet-based histogram is similar to that of the one-dimensional case except that we approximate the *extended cumulative joint distribution* $\mathcal{T}_{1,\dots,d}^{C+}$ instead of \mathcal{T}^{C+} .

In the preprocessing step, we obtain the joint frequency matrix $\mathcal{F}_{1,\dots,d}$ and use it to compute the extended cumulative joint frequency matrix S . We then use the multidimensional wavelet transform to decompose S . There are some common ways to perform multi-dimensional wavelet decomposition. Each of them is a multi-dimensional generalization of the one-dimensional transform described in Section 2.4. In this paper, we use the *standard decomposition*. To obtain the standard decomposition of a multi-dimensional array S , we first apply the one-dimensional transform along one dimension, e.g., dimension X_1 . Next, we apply the one-dimensional transform along the second dimension X_2 on the transform result of the first step, and so on. We repeat this operation until we are done with the last dimension X_d . For example, for two-dimensional case, we first apply the one-dimensional wavelet transform to each row of the two-dimensional array. These transformed rows form a new two-dimensional array. We then apply the one-dimensional transform to each column of the new array. The resulting values form our wavelet coefficients. Af-

ter the decomposition step, pruning is performed to obtain the multi-dimensional wavelet-based histogram H of size m . We can view H as a multi-dimensional array of length m , with each entry being of form $(i_1, i_2, \dots, i_d, v)$, where v is the value of a significant wavelet coefficient and (i_1, i_2, \dots, i_d) is its index.

In the query phase, in order to approximate the selectivity of a range query of the form $(l_1 \leq X_1 \leq h_1) \wedge \dots \wedge (l_d \leq X_d \leq h_d)$, we use the wavelet coefficients to reconstruct the 2^d cumulated counts $S[x_1, x_2, \dots, x_d]$, for $x_j \in \{l_j - 1, h_j\}$, $1 \leq j \leq d$. The following theorem adopted from [HAMS97] can be used to compute an estimate S' for the result size of the range query:

Theorem 2 ([HAMS97]) For each $1 \leq j \leq d$, let

$$s(j) = \begin{cases} 1 & \text{if } x_j = h_j; \\ -1 & \text{if } x_j = l_j - 1. \end{cases}$$

Then the approximate selectivity for the d -dimensional range query specified above is

$$S' = \sum_{\substack{x_j \in \{l_j - 1, h_j\} \\ 1 \leq j \leq d}} \prod_{i=1}^d s(i) \times S[x_1, x_2, \dots, x_d].$$

By convention, we define $S[x_1, x_2, \dots, x_d] = 0$ if $x_j = -1$ for any $1 \leq j \leq d$.

Now we show how to reconstruct the approximate value of $S(x_1, x_2, \dots, x_d)$ using the m wavelet coefficients in our multi-dimensional histogram using the following algorithm:

```

Reconstruct_Value( $H, m, d, x_1, x_2, \dots, x_d$ )
  value = 0;
  for  $j = 1, 2, \dots, m$  do
    if  $Contribute(H[j], i_1, i_2, \dots, i_d)$  then
      value = value +  $Compute_Contribution(H[j], x_1, x_2, \dots, x_d)$ ;
  return value;

```

Function $Contribute(H[j], x_1, x_2, \dots, x_d)$ returns *true* if the j th coefficient $H[j]$ contributes to the reconstruction of the value $S(x_1, x_2, \dots, x_d)$, and it returns *false* otherwise. Function $Compute_Contribution$ computes the actual contribution of $H[j]$ to the reconstruction of the original value $S(x_1, x_2, \dots, x_d)$.

We extend the observations in Section 3 to the multidimensional case and devise two algorithms to compute the multidimensional versions of functions $Contribute$ and $Compute_Contribution$. The CPU time complexity of both algorithms is $O(d)$.

We denote an entry of H by $h = (i_1, i_2, \dots, i_d, v)$ in the following functions.

Function *Contribute*(h, x_1, x_2, \dots, x_d)

//check the coefficient h to see if it contributes to the reconstruction of value $S(x_1, x_2, \dots, x_d)$

contribute = **true**;

$j = 1$;

while (*contribute*) and ($j \leq d$)

//get the the label of the left most leaf for the subtree

//rooted at node i_j in an error tree with $|D_j|$ leaves

$L_l = \text{left_most_leaf}(i_j, |D_j|)$;

//get the the label of the right most leaf for the subtree

//rooted at node i_j in an error tree of $N = |D_j|$

$L_r = \text{right_most_leaf}(i_j, |D_j|)$;

if ($x_j \notin [L_l, L_r]$) //leaf x_j is not in the subtree rooted at node i_j

then *contribute* = **false**;

return *contribute*;

Function *Compute_Contribution*(h, x_1, x_2, \dots, x_d)

//compute the contribution of coefficient h to the reconstruction of value $S(x_1, x_2, \dots, x_d)$

contribution = v ;

for $j = 1, 2, \dots, d$ **do**

if ($i_j \neq 0$) //node i_j is not the root in the error tree of dimension D_j

$depth = \lfloor \log i_j \rfloor$; //compute the depth of node i_j in the error tree

if ($depth < \log |D_{i_j}| - 1$) //node i_j is not a parent of leaves

then

//get the the label of the left most leaf for node i_j 's right

//subtree in an error tree with $|D_j|$ leaves

$L_{rl} = \text{left_most_leaf}(\text{right_child}(i_j), |D_j|)$;

//get the the label of the right most leaf for node i_j 's right

//subtree in an error tree with $|D_j|$ leaves

$L_{rr} = \text{right_most_leaf}(\text{right_child}(i_j), |D_j|)$;

else //node i_j is a parent of leaves

$L_{rl} = \text{right_most_leaf}(i_j, |D_j|)$;

$L_{rr} = L_{rl}$;

//Use Lemma 3

if ($x_j \in [L_{rl}, L_{rr}]$) //leaf x_j is in node i_j 's right subtree

contribution = *contribution* $\times (-1)$;

return *contribution*;

5 Experiments

In this section we report on some experiments that compare the performance of our wavelet-based technique with those of Poosala et al [PIHS96, Poo97a, PI97] and random sampling. Our synthetic data sets are those from previous studies on histogram formation and from the TPC-D benchmark [TPC95]. For simplicity and ease of replication, we use method 1 for pruning in all our wavelet experiments.

5.1 Experimental Comparison of One-Dimensional Methods

In this section, we compare the effectiveness of wavelet-based histograms with MaxDiff(V,A) histograms and random sampling. Poosala et al [PIHS96] characterized the types of histograms in previous studies and proposed new types of histograms. They concluded in their experiments that the MaxDiff(V,A) histograms perform best overall.

Random sampling can be used for selectivity estimation [HS92, HS95, LNS90, LN85]. The simplest way of using random sampling to estimate selectivity is, during the off-line phase, to take a random sample of a certain size (depending on the catalog size limitation) from the relation. When a query is presented in the on-line phase, the query is evaluated against the sample, and the selectivity is estimated in the obvious way: If the result size of the query using a sample of size t is s , the selectivity is estimated as sT/t , where T is the size of the relation.

Our one-dimensional experiments use the many synthetic data distributions described in detail in [PIHS96]. We use $T = 100,000$ to $500,000$ tuples, and the number n of distinct values of the attribute is between 200 and 500.

We use eight different query sets in our experiments:

A: $\{X \leq b \mid b \in D\}$.

B: $\{X \leq b \mid b \in V\}$.

C: $\{a \leq X \leq b \mid a, b \in D, a < b\}$.

D: $\{a \leq X \leq b \mid a, b \in V, a < b\}$.

E: $\{a \leq X \leq b \mid a \in D, b = a + \Delta\}$, where Δ is a positive integer constant.

F: $\{a \leq X \leq b \mid a \in V, b = a + \Delta\}$, where Δ is a positive integer constant.

G: $\{X = b \mid b \in D\}$.

H: $\{X = b \mid b \in V\}$.

Different methods need to store different types of information. For random sampling, we only need to store one number per sample value. The $\text{MaxDiff}(V,A)$ histogram stores three numbers per bucket: the number of distinct attribute values in the bucket, the largest attribute value in the bucket, and the average frequency of the elements in the bucket. Our wavelet-based histograms store two numbers per coefficient: the index of the wavelet coefficient and the value of the coefficient.

In our experiments, all methods are allowed the same amount of storage. The default storage space we use in the experiments is 42 four-byte numbers (to be in line with Poosala et al’s experiments [PIHS96], which we replicate); the limited storage space corresponds to the practice in database management systems to devote only a very small amount of auxiliary space to each relation for selectivity estimation [SAC⁺79]. The 42 numbers correspond to using 14 buckets for the $\text{MaxDiff}(V,A)$ histogram, keeping $m = 21$ wavelet coefficients for wavelet-based histograms, and maintaining a random sample of size 42.

The relative effectiveness of the various methods is fairly constant over a wide variety of value set and frequency set distributions. We present the results from one experiment that illustrates the typical behavior of the methods. In this experiment, the spreads of the value set follow the *cusp_max* distribution with Zipf parameter $z = 1.0$, the frequency set follows a Zipf distribution with parameter $z = 0.5$, and frequencies are randomly assigned to the elements of the value set.³ The value set size is $n = 500$, the domain size is $N = 4096$, and the relation size is $T = 10^5$. Tables 1–5 give the errors of the methods for query sets A, C, E, G, and H. Figure 2 shows how well the methods approximate the cumulative distribution of the underlying data.

Wavelet-based histograms using linear bases perform the best over almost all query sets, data distributions, and error measures. The random sampling method does the worst in most cases. Wavelet-based histograms using Haar bases produce larger errors than $\text{MaxDiff}(V,A)$ histograms in some cases and smaller errors in other cases. The reason for Haar’s lesser performance arises from the limitation of the step function approximation. For example, in the case that both frequency set and value set are uniformly distributed, the cumulative frequency is a linear function of the attribute value; the Haar wavelet histogram produces a sawtooth approximation, as shown

³The *cusp_max* and *cusp_min* distributions are two-sided Zipf distributions. Zipf distributions are described in more detail in [Poo97a]. Zipf parameter $z = 0$ corresponds to a perfectly uniform distribution, and as z increases, the distribution becomes exponentially skewed, with a very large number of small values and a very small number of large values. The distribution for $z = 2$ is already very highly skewed.

in Figure 2b. The Haar estimation can be improved by linearly interpolating across each step of the step function so that the reconstructed frequency is piecewise linear, but doing that type of interpolation after the fact amounts to a histogram similar to the one produced by linear wavelets (see Figure 2a), but without the explicit error optimization done for linear wavelets when choosing the m coefficients.

We also studied the effect of storage space for different methods. Figure 3 plots the results of two sets of our experiments for queries from query set A. In the experiments, the value set size is $n = 500$, the domain size is $N = 4096$, and the relation size is $T = 10^5$. For data set 1, the value set follows *cusp_max* distribution with parameter $z = 1.0$, the frequency set follows a Zipf distribution with parameter $z = 1.0$, and frequencies are assigned to value set in a random way. For data set 2, the value set follows *zipf_dec* distribution with parameter $z = 1.0$, the frequency set follows uniform distribution.

In addition to the above experiments we also tried a modified $\text{MaxDiff}(V, A)$ method so that only two numbers are kept for each bucket instead of three (in particular, not storing the number of distinct values in each bucket), thus allowing 21 buckets per histogram instead of only 14. The accuracy of the estimation was improved. The advantage of the added buckets was somewhat counteracted by less accurate modeling within each bucket. The qualitative results, however, remain the same: The wavelet-based methods are significantly more accurate. Further improvements in the wavelet techniques are certainly possible by quantization and entropy encoding, but they are beyond the scope of this paper.

5.2 Experimental Comparison of Multidimensional Methods

In this section, we evaluate the performance of histograms on two-dimensional (two-attribute) data. We compare our wavelet-based histograms with the $\text{MaxDiff}(V, A)$ histograms computed using the MHIST-2 algorithm [PI97] (which we refer to as MHIST-2 histograms).

In our experiments we use the synthetic data described in [PI97], which is indicative of various real-life data [Bur], and the TPC-D benchmark data [TPC95]. Our query sets are obtained by extending the query sets A–H defined in Section 5.1 to the multidimensional cases.

The main concern of the multidimensional methods is the effectiveness of the histograms in capturing data dependencies. In the synthetic data we used, the degree of the data dependency is controlled by the z value used in generating the Zipf distributed frequency set. A higher z

<i>Error Norm</i>	<i>Linear Wavelets</i>	<i>Haar Wavelets</i>	<i>MaxDiff(V,A)</i>	<i>Random Sampling</i>
$\ e^{\text{rel}}\ _1$	0.6%	4.5%	8%	20%
$\ e^{\text{abs}}\ _{1/T}$	0.16%	0.8%	3%	8%
$\ e^{\text{abs}}\ _{2/T}$	0.26%	0.64%	3.2%	10%
$\ e^{\text{abs}}\ _{\infty/T}$	1.5%	5.6%	11%	13%
$\ e^{\text{comb}}\ _1$, $\alpha = 1, \beta = 100$	0.6	4.4	8	20
$\ e^{\text{comb}}\ _1$, $\alpha = 1, \beta = 1000$	5	30	80	200
$\ e^{\text{comb}}\ _2$, $\alpha = 1, \beta = 1000$	5.1	70.4	12.8	19
$\ e^{\text{comb}}\ _2$, $\alpha = 1, \beta = 1000$	19	224	192	243

Table 1: Errors of various methods for query set A.

<i>Error Norm</i>	<i>Linear Wavelets</i>	<i>Haar Wavelets</i>	<i>MaxDiff(V,A)</i>	<i>Random Sampling</i>
$\ e^{\text{abs}}\ _{1/T}$	0.2%	1.1%	5%	3.5%
$\ e^{\text{abs}}\ _{2/T}$	0.035%	0.18%	0.71%	0.6%
$\ e^{\text{abs}}\ _{\infty/T}$	2.4%	10%	20%	16%

Table 2: Errors of various methods for query set C.

<i>Error Norm</i>	<i>Linear Wavelets</i>	<i>Haar Wavelets</i>	<i>MaxDiff(V,A)</i>	<i>Random Sampling</i>
$\ e^{\text{abs}}\ _{1/T}$	0.1%	0.42%	0.15%	0.35%
$\ e^{\text{abs}}\ _{2/T}$	0.19%	0.96%	0.26%	0.64%
$\ e^{\text{abs}}\ _{\infty/T}$	1.5%	6%	3%	4.6%

Table 3: Errors of various methods for query set E with $\Delta = 10$.

<i>Error Norm</i>	<i>Linear Wavelets</i>	<i>Haar Wavelets</i>	<i>MaxDiff(V,A)</i>	<i>Random Sampling</i>
$\ e^{\text{abs}}\ _{1/T}$	0.03%	0.04%	0.04%	0.04%
$\ e^{\text{abs}}\ _{2/T}$	0.077%	0.32%	0.096%	0.24%
$\ e^{\text{abs}}\ _{\infty/T}$	1.6%	7%	2%	4.6%

Table 4: Errors of various methods for query set G.

<i>Error Norm</i>	<i>Linear Wavelets</i>	<i>Haar Wavelets</i>	<i>MaxDiff(V,A)</i>	<i>Random Sampling</i>
$\ e^{\text{abs}}\ _{1/T}$	0.03%	0.42%	0.2%	0.4%
$\ e^{\text{abs}}\ _{2/T}$	7.7%	16.1%	25.6%	38%
$\ e^{\text{abs}}\ _{\infty/T}$	0.2%	7%	2%	5 %

Table 5: Errors of various methods for query set H.

value corresponds to fewer very high frequencies, implying stronger dependencies between the attributes. One question raised here is what is the reasonable range for that z value. As in [PI97], we fix the relation size T to be 10^6 in our experiments. If we assume our joint value set size is

<i>Data Range</i>	<i>Linear Wavelets</i>	<i>Haar Wavelets</i>	<i>MHIST-2</i>
255	0.3%	1.5%	7%
511	0.3%	1.6%	8%
1023	0.3%	1.6%	6%
2047	0.3%	1.6%	6%

Table 6: $\|e^{\text{abs}}\|_1/T$ errors of various two-dimensional histograms for TPC-D data.

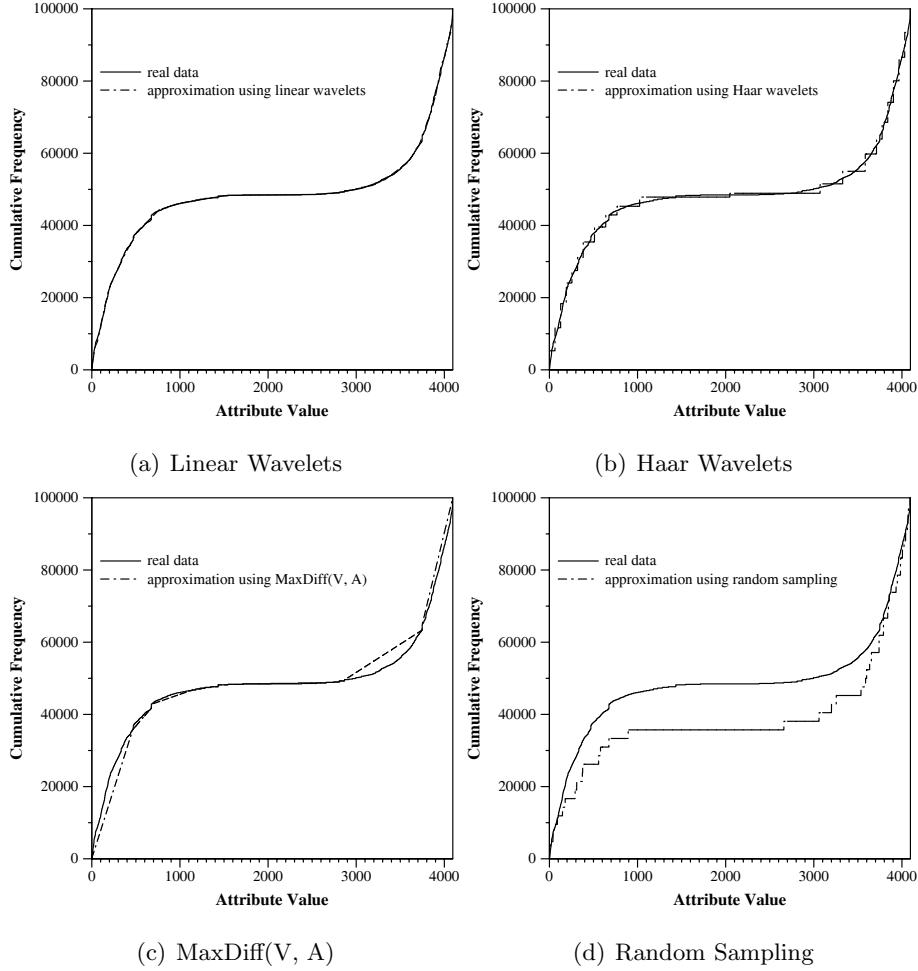


Figure 2: Approximation of the cumulative data distribution using various methods.

$n_1 \times n_2$, then in order to get frequencies that are at least 1, the z value cannot be greater than a certain value. For example, for $n_1 = n_2 = 50$, the upper bound on z is about 1.67. Any larger z value will yield frequency values smaller than 1. In our experiments, we choose various z in the range $0 \leq z \leq 1.5$. The value $z = 1.5$ already corresponds to a highly skewed frequency set; its top three frequencies are 388747, 137443, and 74814, and the 2500th frequency is 3. In [PI97], larger z values are considered; most of the Zipf frequencies are actually very close to 0, so they

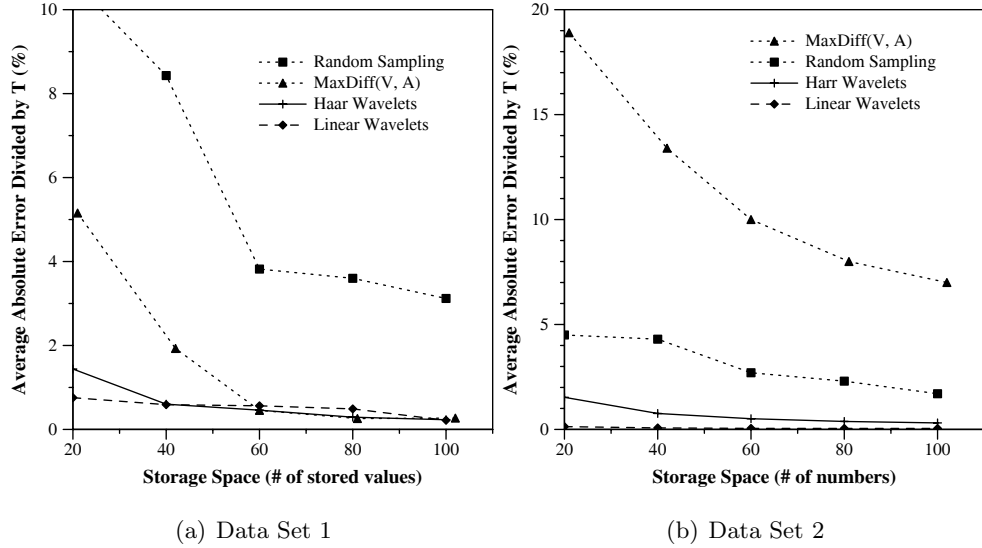


Figure 3: Effect of storage space for various one-dimensional histograms using query set A.

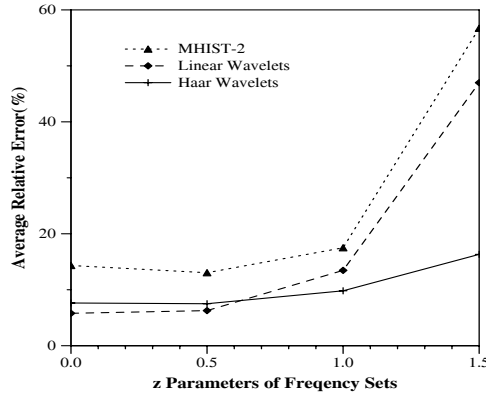


Figure 4: Effect of frequency skew, as reflected by the Zipf z parameter for the frequency set distribution.

are instead boosted up to 1, with the large frequencies correspondingly lowered, thus yielding semi-Zipf distributed frequency sets [Poo97b]. The relative effectiveness of different histograms is fairly constant over a wide variety of data distributions and query sets that we studied. Figure 4 depicts the effect of the Zipf skew parameter z on the accuracy of different types of histograms for one typical set of experiments. In these experiments, we use $N_1 = N_2 = 256$ and $n_1 = n_2 = 50$; the value set in each dimension follows *cusp_max* distribution with $z_s = 1.0$. The storage space is 210 four-bytes numbers (again, to be in line with the default storage space in [PI97]). It corresponds to using 30 buckets for MHIST-2 histogram (seven numbers per bucket) and keeping 70 wavelet coefficients for wavelet-based based histogram (three numbers per coefficient). The queries used

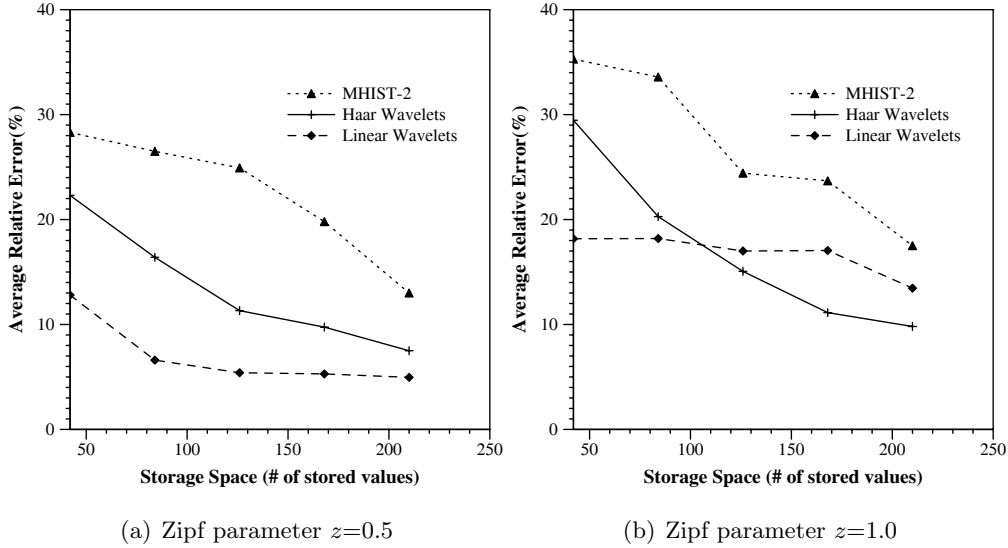


Figure 5: Effect of storage space on two-dimensional histograms.

are those from query set A.

In other experiments we study the effect of the amount of allocated storage space upon the accuracy of various histograms. As we mentioned above, the amount of storage devoted to a catalog is quite limited in any practical DBMS. Even without strict restrictions on the catalog size, a big catalog means that more buckets or coefficients need to be accessed in the on-line phase, which slows down performance. Figure 5 plots the effectiveness of the allocated storage space on the performance of various histograms. In the experiments, we use the same value set and query set as for Figure 4. The frequency skew is $z = 0.5$ for (a) and $z = 1.0$ for (b).

We conducted experiments using TPC-D data [TPC95]. We report the results for a typical experiment here. In this experiment, we use column $L_SHIPDATA$ and column $L_RECEIPTDATA$ in the $LINEITEM$ table, defined as follows:

$$L_SHIPDATA = O_ORDERDATA + random(121)$$

$$L_RECEIPTDATA = L_SHIPDATA + random(30),$$

where $O_ORDERDATA$ is uniformly distributed between values $STARTDATA$ and $ENDDATA - 151$, and $random(k)$ returns a random value between 1 and k . We fix the table size to be $T = 10^6$ and vary the size n of the value set V by means of changing $data\ range$, the difference between $ENDDATA$ and $STARTDATA$. Table 6 shows the $\|e^{abs}\|_1/T$ errors of the different histogram methods for Set A queries.

6 Conclusions

In this paper we have proposed a method to build efficient and effective histograms using wavelet decomposition. Our histograms give improved performance for selectivity estimation compared with random sampling and previous histogram-based approaches. Ever since wavelet transformations were first used for selectivity estimation in the preliminary version of this paper [MVW98], there have been many interesting new developments in using wavelet-based techniques in databases and related applications [Wan99, VWI98, VW99, CGRS00, MVW00, GG02, NRS99, GKMS01, WVLP, DR, PM, MP03, BMW]. We conclude this paper by a brief summary on these new developments.

In [VWI98], a new method based on a logarithm transform is proposed to further reduce the relative errors in wavelet-based approximation. Experiments show that by applying the new method, we can achieve much better accuracy for the data considered in Section 5.

Besides selectivity estimation, an important application area for wavelet-based approximation techniques is On-Line Analytical Processing (OLAP) in data warehouses. Data warehouses can be extremely large, yet obtaining quick answers to queries is important. In many situations, obtaining the exact answer to an OLAP query is prohibitively expensive in terms of time and/or storage space. It can be advantageous to have fast, approximate query answers.

Selectivity estimation is completely internal to the database engine, and the quality (accuracy) of the approximation is observable by a user only indirectly in terms of the performance of the database system. Although a bad (inaccurate) estimation might cause a query to be executed slower, the engine still gives the correct query result. On the other hand, in OLAP applications, approximation plays a more direct role: the query result itself is an approximation. A bad estimation implies a query result with big error that might be unacceptable to a user. Therefore the nature and the quality of approximation become more salient.

Using the wavelet decomposition technique presented in this paper, Vitter and Wang initiate the study of answering OLAP queries approximately in [VW99]. In [VW99], they present novel methods that provide approximate answers to high-dimensional OLAP aggregation queries in massive sparse data sets in a time-efficient and space-efficient manner. The methods consist of I/O-efficient algorithms to construct an approximate and space-efficient representation of the underlying multidimensional data. In the on-line phase, each aggregation query can generally be answered using the compact representation in one I/O or a small number of I/Os, depending

upon the desired accuracy.

While [VW99] mainly addresses multidimensional range-sum queries on a data cube, Chakrabarti et al. extend the use of multi-dimensional wavelets as an effective tool for general-purpose approximate query processing in [CGRS00].

In this paper, we have shown that wavelet-based histograms provide more accurate selectivity estimation answer than random sampling and traditional histogram methods. Since our histograms are built by a nontrivial mathematical procedure, namely, wavelet transformation, it is hard to maintain the accuracy of the wavelet-based histograms when the underlying data distribution changes over time. In particular, simple techniques, such as split and merge, which works well for traditional equi-depth histograms, and updating a fixed set of wavelet coefficients, are not suitable here.

In [MVW00], Matias et al. propose a novel approach based upon probabilistic counting and sampling to maintain wavelet-based histograms and compact data cubes with very little online time and space costs. The accuracy of the method is robust to changing data distributions, and we can get a considerable improvement over previous methods for updating transform-based histograms. A very nice feature of the method is that it can be extended naturally to maintain multidimensional wavelet-based histograms, while traditional multidimensional histograms can be less accurate and prohibitively expensive to build and maintain.

Wavelet-based techniques have also been widely used in data mining [LLZO02] and references therein. In [GKMS01], wavelet technique is used to compute small space representations of massive data streams.

Even though the work in [VW99] and [CGRS00] has demonstrated the effectiveness of wavelet decomposition in reducing large amounts of data to compact set wavelet coefficients that can be used to provide fast and reasonably accurate approximate answers to queries, a major criticism of such techniques is that unlike, for example, random sampling, they do not provide informative error guarantees on the accuracy of individual approximate answers. In [GG02], Garofalakis et al. introduce probabilistic wavelet synopses, the first wavelet-based data reduction technique with guarantees on the accuracy of individual approximate answers.

References

- [BMW] C. Barillon, Y. Matias, and M. Wang. Wavelet-based histograms for selectivity estimation: a systematic study.
- [Bur] U.S. Census Bureau. Census bureau databases. The online data are available on the web at <http://www.census.gov/>.
- [CGRS00] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Approximate query processing using wavelets. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 111–122. Morgan Kaufmann, 2000.
- [Chr83] S. Christodoulakis. Estimating block transfers and join results. In *Proceedings of the 1983 ACM SIGMOD International Conference on Management of Data*, pages 40–50, 1983.
- [Don92] D. L. Donoho. Unconditional bases are optimal bases for data compression and statistical estimation. Technical report, Department of Statistics, Stanford University, 1992.
- [DR] Antonios Deligiannakis and Nick Roussopoulos. Extended wavelets for multiple measures. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, pages 229–240.
- [FM85] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.
- [GG81] E. Gelenbe and D. Gardy. The size of projection of relations satisfying a functional dependency. In *Proceedings of the 1981 International Conference on Very Large Databases*, pages 325–333, 1981.
- [GG02] Minos N. Garofalakis and Phillip B. Gibbons. Wavelet synopses with error guarantees. In *SIGMOD 2002, Proceedings ACM SIGMOD International Conference on Management of Data, June 3-6, 2002, Madison, Wisconsin, USA*. ACM Press, 2002.
- [GKMS01] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In

VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy. Morgan Kaufmann, 2001.

- [GM98] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, WA, June 1998.
- [GMP97] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of the 1997 International Conference on Very Large Databases*, Athens, Greece, August 1997.
- [HAMS97] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, AZ, May 1997.
- [HS92] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, 1992.
- [HS95] P. Haas and A. Swami. Sampling-based selectivity for joins using augmented frequent value statistics. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, March 1995.
- [LLZO02] T. Li, Q. Li, S. Zhu, and M. Ogihara. A survey on wavelet applications in data mining. *SIGKDD Explorations*, 4(2):49–68, 2002.
- [LN85] R. Lipton and J. Naughton. Query size estimation by adaptive sampling. *J. of Comput. Sys. Sci.*, 51:18–25, 1985.
- [LNS90] R. Lipton, J. Naughton, and D. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceeding of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 1–11, 1990.
- [MD88] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 28–36, 1988.
- [MP03] Y. Matias and L. Portman. Workload-based wavelet synopses. Technical report, 2003.
- [MVW98] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity es-

- timization. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 448–459, Seattle, WA, June 1998.
- [MVW00] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 101–110. Morgan Kaufmann, 2000.
- [NRS99] Apostol Natsev, Rajeev Rastogi, and Kyuseok Shim. Walrus: A similarity retrieval algorithm for image databases. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 395–406. ACM Press, 1999.
- [PI97] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 1997 International Conference on Very Large Databases*, Athens, Greece, August 1997.
- [PIHS96] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, May 1996.
- [PM] Ivan Popivanov and Renee J. Miller. Similarity search over time-series data using wavelets. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02) February 26 - March 01, 2002 San Jose, California*, pages 212–224.
- [Poo97a] V. Poosala. *Histogram-Based Estimation Techniques in Database Systems*. Ph. D. dissertation, University of Wisconsin-Madison, 1997.
- [Poo97b] V. Poosala. Personal communication, 1997.
- [PSC84] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 256–276, 1984.
- [SAC⁺79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pages 23–34, 1979.

- [SDNR96] A. Shukla, P. Deshpande, J. F. Naughton, and K. Ramaswamy. Storage estimation for multidimensional aggregates in the presence of hierarchies. In *Proceedings of the 1996 International Conference on Very Large Databases*, pages 522–531, 1996.
- [SLRD83] W. Sun, Y. Ling, N. Rishe, and Yi Deng. An instant and accurate size estimation method for joins and selections in a retrieval-intensive environment. In *Proceedings of the 1983 ACM SIGMOD International Conference on Management of Data*, pages 79–88, 1983.
- [TPC95] TPC benchmark D (decision support), 1995.
- [Vit85] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, March 1985.
- [Vit87] J. S. Vitter. An efficient algorithm for sequential random sampling. *ACM Transactions on Mathematical Software*, 13(1):58–67, March 1987.
- [Vit01] J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2):207–271, June 2001.
- [VW99] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 193–204, Philadelphia, June 1999.
- [VWI98] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proceedings of Seventh International Conference on Information and Knowledge Management*, pages 96–104, Washington D.C., November 1998.
- [Wan99] Min Wang. *Approximation and Learning Techniques in Database Systems*. Ph. D. dissertation, Duke University, 1999.
- [WVLP] Min Wang, Jeffrey Scott Vitter, Lipyeow Lim, and Sriram Padmanabhan. Wavelet-based cost estimation for spatial queries. In *Advances in Spatial and Temporal Databases : Proceedings of the 7th International Symposium, SSTD 2001, Redondo Beach, CA, USA, July 12-15, 2001*, pages 175–196. Springer-Verlag Heidelberg.