# Fast Algorithms For Hierarchical Range Histogram Construction

**Sudipto Guha**
UPenn
sudipto@cis.upenn.edu

**Nick Koudas**
AT&T Labs–Research
koudas@research.att.com

**Divesh Srivastava**
AT&T Labs–Research
divesh@research.att.com

## ABSTRACT

Data Warehousing and OLAP applications typically view data as having multiple logical dimensions (e.g., product, location) with natural hierarchies defined on each dimension. OLAP queries usually involve hierarchical selections on some of the dimensions, and often aggregate measure attributes (e.g., sales, volume). Accurately estimating the distribution of measure attributes, under hierarchical selections, is important in a variety of scenarios, including approximate query evaluation and cost-based optimization of queries.

In this paper, we propose fast (near linear time) algorithms for the problem of approximating the distribution of measure attributes with hierarchies defined on them, using histograms. Our algorithms are based on dynamic programming and a novel notion of sparse intervals that we introduce, and are the first *practical* algorithms for this problem. They effectively trade space for construction time without compromising histogram accuracy. We complement our analytical contributions with an experimental evaluation using real data sets, demonstrating the superiority of our approach.

## 1. INTRODUCTION

Data warehousing and OLAP applications have firmly established themselves as crucial to today's businesses. OLAP applications typically view data as having multiple logical dimensions (e.g., product, location) with natural hierarchies defined on each dimension, and analyze the behavior of various measure attributes (e.g., sales, volume) in terms of the dimensions. OLAP queries usually involve hierarchical selections on some of the dimensions (e.g., product is classified under the `jeans` product category, or location is in the `north-east` region), and often aggregate measure attributes (see, e.g., [6]). Accurately estimating the distribution of measure attributes, under hierarchical selections, is important in a variety of scenarios, including approximate OLAP query evaluation and cost-based optimization of OLAP queries.

Histograms capture attribute value distribution statistics in a space-efficient fashion. They have been designed to work well for numeric attribute value domains, and have long been used to support cost-based query optimization in databases [12, 10, 2, 4, 11, 5]. Histograms can be used to estimate the selectivity of OLAP queries by modeling the (hierarchical) conditions on a given dimension as a set of *hierarchical ranges* (i.e., two ranges are either disjoint or one is contained in the other), and using standard range selectivity estimation techniques (see, e.g., [11]).

The quality of selectivity estimates obtained using a histogram depends on computing a good solution to the histogram construction problem, and there has been considerable recent effort in this area (see, e.g., [11, 5, 7]). For the most part, previous works on computing good histograms (see, e.g., [11, 5]) consider only equality queries when computing the error incurred by a particular choice of histogram bucket boundaries. Koudas et al. [7] showed that using histograms optimal for equality queries can result in sub-optimal estimates for hierarchical range selections.

To address this problem, [7] presented a polynomial-time, dynamic programming algorithm for computing histograms that provably minimizes expected error for a given number of buckets, for the case of arbitrary hierarchical range selections. The high complexity of this algorithm ($O(|T|n^6B^2)$, where $T$ is the size of the hierarchy, $n$ is the number of leaf-level values and $B$ is the


*ACM PODS* '2002 June 3-6, Madison, Wisconsin, USA

number of buckets) makes it quite impractical, even for moderate sized datasets.

In this paper, we propose fast, practical algorithms for the problem of constructing hierarchical range histograms to approximate the distribution of measure attributes with hierarchies defined on them. In particular, we make the following contributions:

- We introduce a novel notion of sparse intervals (Section 3), and use it to formulate a family of dynamic programming algorithms that construct hierarchical range histograms (Section 4).

  Our algorithms effectively trade space (number of buckets) for construction time without compromising histogram accuracy. In particular, one can devise fast (near linear time) algorithms for our problem.

- We complement our analytical contributions with an experimental evaluation using real data sets, comparing our fast algorithm with previous proposed heuristics for arbitrary range histograms (Section 5).

  We demonstrate the superiority of our technique, both in the quality of the histograms constructed, and the time taken to construct them.

Our work provides the *first* practical approach to the problem of constructing hierarchical range histograms, and solves a key open problem in this area.

## 1.1 Related Work

Histogram construction techniques and their use in selectivity estimation for query optimization as well as their relationship to approximate query answering, have a long research history in the database literature [8, 3, 2, 4, 11, 5, 9]. The V-Optimal histogram was defined in [2, 4] to be that which minimizes error for estimating equality queries. Jagadish et al., [5] proposed an optimal polynomial-time algorithm to construct V-Optimal histograms. Heuristically constructed V-Optimal histograms were evaluated in [11] for range selection predicates along with several other histograms and were shown to achieve good accuracy for those predicates as well. However, the evaluation of V-Optimal histograms for range queries was performed on V-Optimal histograms (or approximations thereof) constructed by taking only equality queries into account.

Most closely related to our work is that of Koudas et al. [7], who identified the need for hierarchical range histograms, and presented polynomial-time, dynamic programming algorithms for computing histograms that provably minimize expected error for a given number of buckets, for the special cases of prefix ranges and balanced binary trees, as well as for the general case of arbitrary hierarchical range selections. More recently, Gilbert et al. [1] presented pseudopolynomial-time histogram construction algorithms that are optimal or provably approximate for arbitrary range queries. The high polynomiality of these algorithms is a principal motivation for our work.

# 2. PROBLEM DEFINITION

We are given an array $A[1,n]$ of non-negative real numbers. Define $\overline{A[a,b]} = \frac{A[a]+\cdots+A[b]}{b-a+1}$, the average of items $A[a],\ldots,A[b]$. Intuitively, a histogram of array $A$ using $B$ buckets is a disjoint partition of the range $[1,n]$ into $B$ intervals. More formally, we have the following definition:

Definition 2.1. **[Histogram]** *A* histogram *of array $A[1,n]$ using $B$ buckets is specified by $B+1$ integers, $b_1,\ldots,b_{B+1}$, where $0=b_1 \leq b_2 \leq \ldots \leq b_{B+1}=n$. Each interval $[b_i+1, b_{i+1}]$ is called a* bucket, *and each $b_i$ is called a* bucket boundary. ∎

The histogram is stored as the series of bucket boundaries together with the average of the array values in each bucket, i.e., $\overline{A[b_i+1,b_{i+1}]}$. This implies that for any bucket $[b_i+1,b_{i+1}]$, we can obtain the sum of all values in it from the average and its length, both of which are available from the histogram representation.

For a given array $A$, many histograms are possible, as discussed in Section 1.1. Most previous works on histograms make use only of equality queries of the form "*give me* $A[i]$" in their definition and construction. Here, we use the definition of "optimal" histograms for a more general query workload consisting of hierarchical range queries, from [7].

Definition 2.2. **[Hierarchical Range Queries]** *A range query $R_{ij}$ asks for the sum of the values $s_{ij} = A[i]+\cdots+A[j]$. A set $S$ of range queries is said to be* hierarchical, *if for any two queries $R_{ij}$ and $R_{k\ell}$ in $S$, either the ranges $[i,j]$ and $[k,\ell]$ are disjoint, or one is contained in the other.* ∎

Hierarchical range queries generalize equality queries since the degenerate range query $R_{ii}$ is precisely the equality query $A[i]$. Hierarchical range queries can be conveniently displayed as a tree in which each node $u$ has a range $r_u$ associated with it. Node $v$ is a child of node $u$ if and only if $r_v$ is contained in $r_u$, but there is no other node $w$ such that $r_w$ contains $r_v$ and is contained in $r_u$.

We now define the notion of a query workload, based on a simple model of probabilistic query distribution, where the probability of any particular range query being asked is independent of the probabilities of other range queries.

DEFINITION 2.3. **[Workload]** *A workload $W$ consists of a set $S$ of hierarchical range queries, along with a probability $p_{ij}$ associated with each range query $R_{ij}$ in $S$.* ∎

The probability $p_{ij}$ associated with every range can be obtained by monitoring and logging queries on the warehouse.

Given a histogram $H$ of array $A[1,n]$, a range query $R_{ij}$ is answered as follows. Recall that the answer is $s_{ij} = A[i] + \cdots + A[j]$. We consider the *left* bucket $[b_\ell + 1, b_{\ell+1}]$ that "straddles" $i$, i.e., $b_\ell + 1 \le i \le b_{\ell+1}$. Likewise, the *right* bucket $[b_r + 1, b_{r+1}]$ is defined as the one that straddles $j$. Thus, the range of query $R_{ij}$ contains portions of the left and right buckets, and contains every bucket between these two in its entirety. (The left and right buckets may coincide, and/or there may be no buckets in between, but our discussion here is not seriously affected.) We can obtain the precise total of all values in the buckets between the left and right buckets from the histogram representation, as remarked earlier. For the portions within the left and right buckets, we use the common assumption of *uniformity*, i.e., we estimate the sum of the $A$ values in the interval $[i,j] \cap [b_\ell+1, b_{\ell+1}]$ as $\overline{A[b_\ell+1, b_{\ell+1}]} * |[i,j] \cap [b_\ell+1, b_{\ell+1}]|$, and likewise for the right bucket. The total estimate for $s_{ij}$ is then the sum of the estimates for the left and right buckets, and the exact sums for the buckets in between. We denote this estimate by $\hat{s}_{ij}$.

Optimal histograms are defined based on the errors in the estimation for queries [7].

DEFINITION 2.4. **[Optimal Histogram]** *Given a histogram $H$ of array $A[1,n]$, the* error $e_{ij}$ *of range query $R_{ij}$ is defined to be $(s_{ij} - \hat{s}_{ij})^2$.*

*Given a histogram $H$ of array $A[1,n]$, and a workload $W$, the* total expected error *for estimating $W$ is defined to be $\sum_{R_{ij}} (p_{ij} e_{ij})$, over all queries $R_{ij}$ in $W$.*[1]

*Given a workload $W$, an* optimal histogram with $B$ buckets *of array $A[1,n]$ is the histogram with at most $B$ buckets that has the minimum total expected error for estimating workload $W$, among all histograms with at most $B$ buckets.* ∎

Koudas et al. [7] showed that constructing the optimal histogram for hierarchical range queries requires a high polynomial time (in $n$), which makes it unsuitable in practice. As a result, we seek faster algorithms that trade space (number of buckets) with construction time, for a specified error. We can now formally state the problem that we address in this paper:

> **Fast Histogram (FH) Construction for Hierarchical Range Queries**: Given array $A[1,n]$, $B$ buckets, and a workload $W$ of hierarchical range queries with probability $p_{ij}$ for query $R_{ij}$, let $E$ denote the total expected error of the optimal hierarchical range histogram with $B$ buckets. We seek algorithms that construct hierarchical range histograms (boundaries and bucket averages) with an error at most $E$, trading space with construction time.

There are some variations on the basic problem described above that are of interest in general. For example, a range query $R_{ij}$ may return not the sum of the $A[k]$ values in that range, but say the maximum $\max_{k=i}^{j} A[k]$, or any other suitable function. Another variation is that one may choose a different representation for the histogram, say, by storing not the average value, but some other representative(s) for the bucket values. One possibility would be to store a value that optimizes the error over the range queries. Our discussions below will hold for many such variations of optimal histograms, but we do not elaborate on this issue further.

## 3. A SPARSE INTERVAL SET SYSTEM

As a preliminary to our algorithm for constructing hierarchical range histograms, we investigate a notion of a *sparse interval set* system that allows us to conceptually partition the algorithm for histogram construction into two steps. In the first step, we will construct a set of "sparse" intervals to which we would be interested in restricting our computation. We will show that this process may increase the number of buckets used but will be able to represent any arbitrary interval in a desirable form involving only intervals from the sparse set. The dynamic programming algorithm will be performed in the second step.

Consider an integer $r \ge 1$ as a parameter and set $\ell = n^{\frac{1}{r}} > 1$. Consider the following collection of points:

- Without loss of generality assume the items are indexed as $0, \ldots, n$. These are defined as level 1 points.

[1]This is the standard sum-squared-error measure.

- Consider the numbers $0, \ell, 2\ell, 3\ell, .....$ These are defined as level 2 points.

- In general, the sequence $0, \ell^j, 2\ell^j, 3\ell^j, \ldots$ are defined as level $j+1$ points.

- Overall we have $r+1$ levels. The highest level points are $0, n$.

- The interval $[0, n]$ is in the sparse system $S$. Any pair of level $j$ points between adjacent level $j+1$ points defines an interval in $S$.

It is not hard to verify that any interval over $[0, n]$ can be written as a disjoint union of at most $2r$ intervals in the sparse system. To show that, it will be sufficient to show the following result:

Claim 3.1. *Any interval $[0, x]$ can be expressed as a partition of at most r intervals from the above collection.*

Proof. We will show by induction on $x$ that any interval $[0, x]$ where $x \le \ell^j$ can be expressed as $j$ intervals. Notice it is sufficient to prove the result for minimal $j$ such that $x \le \ell^j$. Assume the result to be true for $x \le \ell^j$, that is $[0, x]$ can be expressed as $j$ intervals. This is true for $j=1$, the base case.

Now consider $\ell^j < x \le \ell^{j+1}$. We can write the interval as $[0, t\ell^j]$ and $[t\ell^j, x]$ where $t$ is maximal. Observe that $t \le \ell$. Thus the interval $[0, t\ell^j]$ is a valid pair of points between adjacent level $j+1$ points 0 and $\ell^{j+1}$. Thus we would use one interval to represent the former, and the latter is essentially similar to interval $[0, x - t\ell^j]$. Since $t$ is maximal, $x - t\ell^j \le \ell^j$. Therefore the latter interval can be expressed as a disjoint union of at most $j$ intervals. Thus by induction any interval $[0, x]$ can be expressed as a disjoint union of at most $j$ intervals where $j$ is the smallest integer such that $x \le \ell^j$. As mentioned, as a corollary, any interval $[0, x]$ can be expressed as a union of $r$ intervals, since $x \le \ell^r$. ∎

Observe that any interval $[a, b]$ can be expressed as $2r$ intervals by cutting it in a suitable point of the form $a \le \alpha\ell^j \le b$ with maximum $j$. Then, by symmetry, each side $[a, \alpha\ell^j]$ and $[\alpha\ell^j, b]$ can be expressed as a disjoint union.

We can actually claim the following interesting fact:

Lemma 3.1. *In a sparse set system with parameter $r$, the number of intervals containing a point is at most $O(rn^{\frac{2}{r}})$.*

Proof. Consider the level 1 intervals. There can be at most $O(\ell^2)$ of these that contain a specific point, because the start and end points are bound within adjacent level 2 points that contain this point. Since the level $j$ intervals behave the same way on level $j$ points as the level 1 intervals behave on any of the original points, there are $O(\ell^2)$ level $j$ intervals that include the given point. Extending this argument over the $r$ levels, the claim follows. The $r+1$'th level adds only one interval. ∎

# 4. THE DYNAMIC PROGRAMMING ALGORITHM

Consider the array $A[1, n]$ with bucket boundaries $0, \ldots, n$. Recall that the bucket with boundaries $b_i$ and $b_{i+1}$ corresponds to the interval $[b_i + 1, b_{i+1}]$, and the value stored in a histogram bucket is the average of the values in the interval. Given $b_i$ and $b_{i+1}$, this value can be determined in constant time if we construct a prefix sum array, $\sum_{j=1}^{b_i} A[j]$ for all $b_i$.

Computing the estimate of a range $R_{ij}$ is described in a previous section. Let us denote the estimate of range $R_{ij}$ by $\hat{s}_{ij}$, the true sum of range $R_{ij}$ by $s_{ij}$, and the error by $e_{ij}$.

$$e_{ij} = (s_{ij} - \hat{s}_{ij})^2$$

We first start with the observation that given a range $R_{ij}$, any bucket that is completely contained in $R_{ij}$ does not contribute to the error in estimation of $R_{ij}$. In other words, the error of $R_{ij}$ remains the same if the values in this bucket did not exist. Therefore, if we are interested in estimating the error of a range, we are interested only in (at most) two buckets that overlap partially with the range, one at either end. Once these two buckets are fixed, the error of the range is fixed. Further, buckets that are completely contained in $R_{ij}$ do not affect the error of any range disjoint from $R_{ij}$. These completely contained buckets may, however, affect the error of subranges of $R_{ij}$. This sets the stage for a dynamic programming approach. Given range $R_{ij}$, the two buckets that partially overlap $R_{ij}$ at either end, and the number of buckets completely contained in $R_{ij}$, the algorithm finds the best way to place these buckets inside $R_{ij}$.

Before describing the dynamic programming algorithm, recall that the ranges define a hierarchy, based on the inclusion relationship between the ranges. This hierarchy can be represented by a tree, denoted by $T$. With each node $v$ of the tree $T$, a range $R_{ij} = [v_L, v_R]$ is associated where $i = v_L$ and $j = v_R$. The weight $w_v$ of a node $v$ is $p_{ij}$, and the error $e_v$ is $e_{ij}$.

If we allow $w_u = 0$ for some nodes $u$ in the tree, then we can easily assume that the tree is binary, with at most two children of any node. Assuming that a node $v$ in the tree had children $u_1, \ldots, u_t$, we can easily transform this node into a new node $v$ that has two children in the new tree, the first corresponding to $u_1$ and the other corresponding to $\langle u_2, \ldots, u_t \rangle$ with weight 0. It is easy to note that the size of the tree increases only by a factor of 2.

Now we are ready to formulate the dynamic programming algorithm $FH$. Let $\texttt{Best}(v, left, right, p)$ denote the smallest error of the range $[v_L, v_R]$ denoted by node $v$ of the tree with the (possibly partial) overlapping interval on the left being $left$ and the partial overlapping interval on the right being $right$, and $v$ contains $p$ intervals completely. Formally, $left$ contains $v_L$ and $right$ contains $v_R$. We use $left = \Phi$ to indicate that $v_L$ is the starting point of an interval. Similarly for $right$.

- Let the children of $v$ be $y$ and $z$. Let the ranges denoted by these nodes be $[y_L, y_R]$ and $[z_L, z_R]$.
- Case (a): For all possible intervals $I$ that contain $y_R$ and $z_L$ (see Figure 1(a)), compute
$$cost(I) = \min_{k_1} w_y e_y + w_z e_z + \texttt{Best}(y, left, I, p - k_1 - 1)$$
$$+\texttt{Best}(z, I, right, k_1)$$
- Case (b): In the case that $I$ finishes on $y_R$ (see Figure 1(b)),
$$cost(I) = \min_{k_1} w_y e_y + w_z e_z + \texttt{Best}(y, left, I, p - k_1)$$
$$+\texttt{Best}(z, \Phi, right, k_1)$$
- Note that when the interval $I$ is fixed, $e_y$ and $e_z$ are automatically defined and can be computed in $O(1)$ time.
- Return $\min_I cost(I)$.

It is straightforward to observe that the time spent in evaluating $cost(I)$ is $O(p)$. The running time of the above algorithm therefore depends on the number of choices of interval $I$. Let $C(S)$ be the maximum number of intervals in an interval system $S$ that contain a particular element (in our case $y_R$); if all intervals are allowed, $C(S) = O(n^2)$. Thus, the running time of Algorithm $FH$ is $O(pC(S))$ where the solution is a subcollection of the set of intervals of $S$. Now, the number of entries for each tree node $v$ is $O(BC(S)^2)$. This is because there are $C(S)+1$ intervals for choices of $left$ (all intervals that contain $v_L$ and $\Phi$). Similarly for $right$.

Thus, the work for every tree node is $O(B^2C(S)^3)$, and the total work including preprocessing is
$$O(n + |T|B^2C(S)^3)$$
When $S$ is the set of all possible intervals, this gives a $O(n^6B^2|T|)$ algorithm, which matches the time complexity of the algorithm of [7] (even though the dynamic programming formulations are quite different). However, if $S$ is a sparse set system of parameter $r$, $C(S) = rn^{\frac{2}{r}}$. In this case, the solution may now be represented as $2r(B-1)$ intervals[2].

What we perform is the dynamic programming algorithm mentioned above, with the number of buckets as $2r(B-1)$. Using the results of the previous section, since a histogram with $B$ buckets can be expressed as a histogram with $2r(B-1)$ buckets in the sparse system, the solution that we will get from the dynamic program will have error less than or equal to the original $B$ bucket histogram with arbitrary intervals. Setting $\epsilon = \frac{6}{r} \leq 6$ we can claim the following,

THEOREM 4.1. *In time $O(n + B^2\epsilon^{-5}n^{\epsilon}|T|)$ we can construct a solution with $O(B/\epsilon)$ buckets whose error is at most the error of any solution of the original problem with $B$ buckets.* ∎

It is possible to get alternate tradeoffs than mentioned in the above theorem by constructing different sparse set systems. For instance, if we construct the complete binary tree on $[0, n]$ and allow intervals such that one end point is an ancestor of the other in the complete binary tree, we can show that any arbitrary interval in $[0, n]$ can be expressed as a disjoint union of two intervals from the sparse set. In this case, $C(s) = O(n)$, and we would get a solution with $2B$ buckets in time $O(n^3|T|B^2)$. We relegate further discussion of possible different tradeoffs using alternate constructions to the full paper.

## 5. EXPERIMENTAL EVALUATION

To assess the performance of the proposed $FH$ algorithm (the instance of the family of dynamic programming algorithms with $r = 6$), we implemented it. In this section, we present an evaluation of its accuracy and performance. Recently, Gilbert et al. [1] presented algorithms for histogram construction with good properties for arbitrary range queries. They presented a series of computationally expensive algorithms to construct provably good histograms for arbitrary range queries

[2]Notice that also increases the running time by a factor of $r^2$ or $O(\frac{1}{\epsilon^2})$.

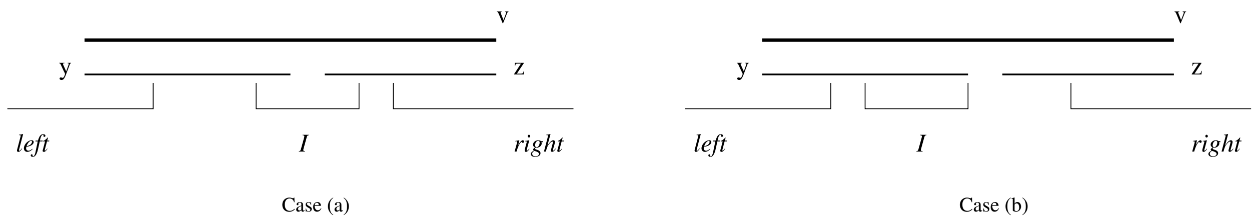



**Figure 1: Cases for Algorithm $FH$**

(assuming all range queries are equally likely); however, workload information cannot easily be folded into these algorithms. The most practical algorithm Gilbert et al. discuss, having relatively low polynomial complexity, is named $A0$. All the other algorithms have very high polynomial complexity and cannot scale to large data collections. For a data series of length $n$ to be approximated with $B$ buckets the $A0$ algorithm has complexity $O(n^2B)$ and constructs a histogram consisting of $2B$ buckets. Moreover, $A0$ was experimentally evaluated in [1] and shown to have very good accuracy. Since $A0$ is the only known algorithm of reasonable complexity optimized for arbitrary range queries, we choose to evaluate our $FH$ algorithm against $A0$.

**Description of Data Sets and Workloads:.** We used both real and synthetic data sets in our experiments. In particular we experimented with the following data sets:

- `A`: A real data set of length 1000 extracted from an AT&T operational warehouse.
- `B`: A synthetic data set of length 2000, distributed according to the Zipf distribution with skew parameter 0.5.
- `C`: A synthetic data set (of varying length), representing samples from a Gaussian distribution with mean and variance 250.

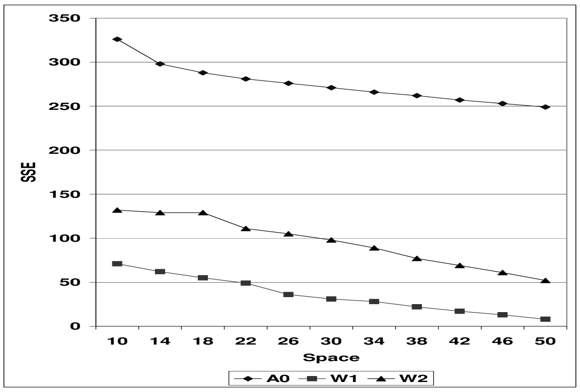
We obtain the query distribution for the hierarchical queries using a Gaussian distribution and vary the variance in order to observe the performance trends. In particular we use a normal to assign the probabilities to a full hierarchy and normalize to obtain a probability distribution. Then we generate the workload queries according to this distribution. We experiment with two workloads `W1`(generated by sampling $N(10,10)$) and `W2` (generated by sampling $N(10,50)$). We compute the accuracy by asking 1000 queries derived according to the corresponding query distribution and we report the total (expected) sum squared error of the workload execution on the histograms.

**Performance Evaluation:.** Our experiments seek to quantify both the accuracy and the construction time for our algorithm, as parameters of interest vary. These parameters include the total space allowed for histogram approximation as well as the total size of the data set that we approximate using the histograms.

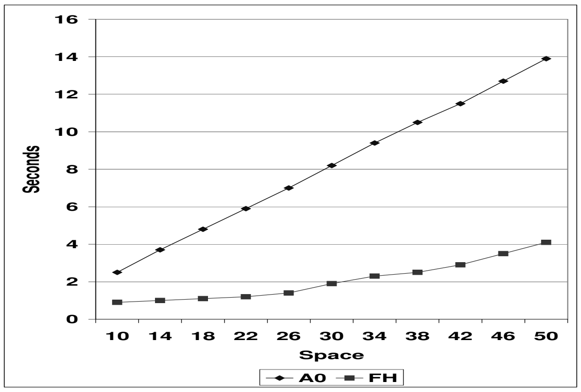
Figure 2(a) presents the accuracy of $FH$ versus that of $A0$ as the total space allowed for histogram approximation increases for data set `A`. For the case of algorithm $FH$ we report results using workloads `W1` and `W2`. It is evident that the accuracy of the $FH$ histogram is superior to that of $A0$. For smaller variance (workload `W1`), the $FH$ histogram appears more accurate. As the variance increases, the query distribution gets closer to uniform, which is what $A0$ histograms are optimized for. Figure 2(b) presents the construction time for both histograms. Since we vary the total space (and implicitly the number of buckets), keeping the data size fixed, construction time for $A0$ appears linear in the space, as expected from the analytical expression of its construction complexity. For $FH$ histograms, the value of the error changes implicitly to utilize the specific space budget and as is evident from the figure, the savings in construction time are substantial for the same range of space allowed to both histograms.

Figures 3(a)(b) present the results of the same experiment for data set `B`. The observations are similar as in the case of data set `A`. Accuracy improves much faster with space in this case, since the distribution is Zipf; as soon as the leading terms are approximated with buckets, the remainder of the distribution consists of small values incurring small error after approximation. Notice, however, that the savings in construction time in the case of $FH$ are dramatic since data set `B` is twice the size of `A`.

Figure 4(a) presents the accuracy of $FH$ compared to that of $A0$ as the data set size increases for data set `C`, and total space 20. Trends in accuracy are consistent for $FH$. For $A0$ the curve appears to have a plateau, and this is due to the way the data set is generated and used in the experiment. Essentially as the size in-

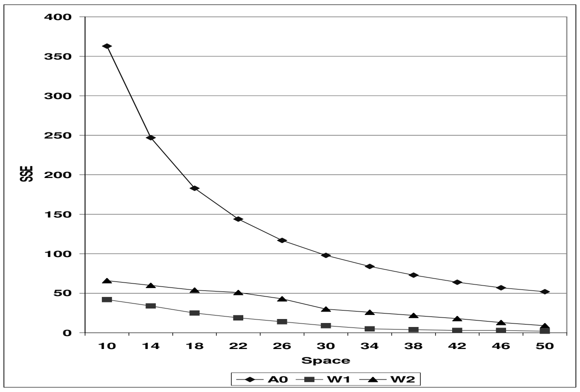


(a) Accuracy (SSE)

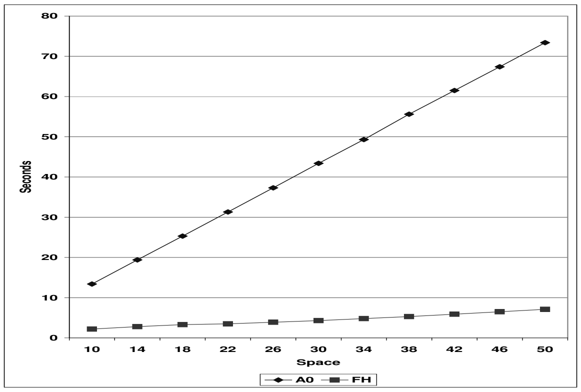


(b) Construction Time

**Figure 2: Results for data set** `A` **as total space allowed for histogramming increases**

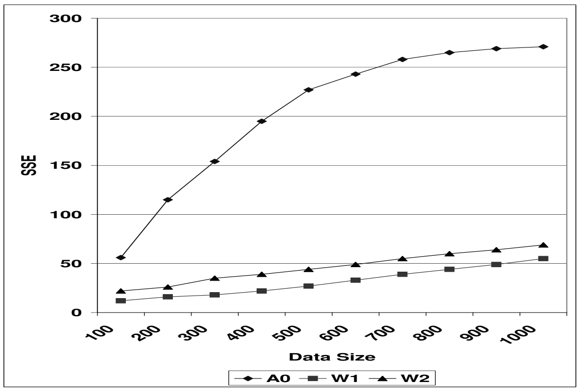


(a) Accuracy (SSE)

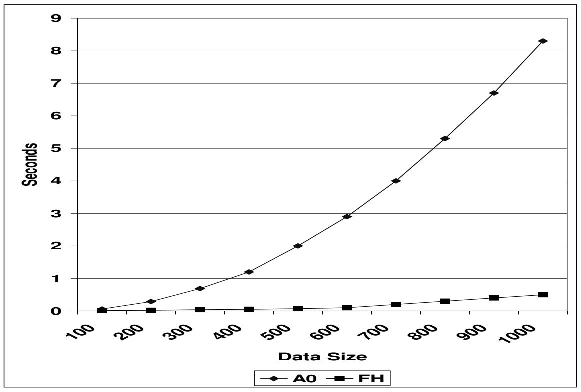


(b) Construction Time

**Figure 3: Results for data set** `B` **as total space allowed for histogramming increases**



(a) Accuracy (SSE)



(b) Construction Time

**Figure 4: Results for data set** `C` **as data set size increases**

creases, the tail of the Gaussian is considered, and the approximation error is very small. Figure 4(b) presents construction time as the data set size increases for data set C. The trends are consistent with our analytical expectations, with $A0$ showing a clear quadratic trend in construction time and $FH$'s construction time increasing near-linearly with increasing data set size.

## 6. CONCLUSIONS

We have presented the first practical approach to the problem of constructing hierarchical range histograms. Our dynamic programming algorithms effectively trade space for construction time without compromising histogram accuracy. They are based on a novel notion of sparse intervals, which are of independent interest, and likely to find applications in other approximation scenarios.

An important open direction for future work is a formal study of the dynamic properties of hierarchical range histograms: how should one modify these histograms under data or workload modifications?

## 7. REFERENCES


[1] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Optimal and approximate computation of summary statistics for range aggregates. In *Proceedings of the ACM Symposium on Principles of Database Systems*, 2001.

[2] Y. Ioannidis. Universality of serial histograms. In *Proceedings of the International Conference on Very Large Databases*, pages 256–267, 1993.

[3] Y. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. *ACM Trans. Database Syst.*, 18(4):709–748, Dec. 1993.

[4] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 233–244, 1995.

[5] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proceedings of the International Conference on Very Large Databases*, pages 275–286, 1998.

[6] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses? In *Proceedings of the International Conference on Very Large Databases*, Edinburgh, Scotland, UK, Sept. 1999.

[7] N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal histograms for hierarchical range queries. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 196–204, 2000.

[8] M. V. Mannino, P. Chu, and T. Sager. Statistical profile estimation in database sysems. *ACM Computing Surveys*, 20(3):191–221, Sept. 1988.

[9] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 448–459, 1998.

[10] M. Muralikrishna and D. Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 28–36, 1988.

[11] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range queries. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 294–305, 1996.

[12] P. G. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access path selection in a relational database management system. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, June 1979.