A Fast Approximation Scheme for Probabilistic Wavelet Synopses

Antonios Deligiannakis
University of Maryland
adeli@cs.umd.edu

Minos Garofalakis
Bell Laboratories
minos@research.bell-labs.com

Nick Roussopoulos
University of Maryland
nick@cs.umd.edu

December 9, 2004

Abstract

Several studies have demonstrated the effectiveness of Haar wavelets in reducing large amounts of data down to compact *wavelet synopses* that can be used to obtain fast, accurate approximate query answers. While Haar wavelets were originally designed for minimizing the overall root-mean-squared (i.e., L_2-norm) error in the data approximation, the recently-proposed idea of *probabilistic wavelet synopses* also enables their use in minimizing other error metrics, such as the relative error in individual data-value reconstruction, which is arguably the most important for approximate query processing. Known construction algorithms for probabilistic wavelet synopses employ probabilistic schemes for coefficient thresholding that are based on optimal Dynamic-Programming (DP) formulations over the error-tree structure for Haar coefficients. Unfortunately, these (exact) schemes can scale quite poorly for large data-domain and synopsis sizes. To address this shortcoming, in this paper, we introduce a novel, fast *approximation scheme* for building probabilistic wavelet synopses over large data sets. Our algorithm's running time is near-linear in the size of the data-domain (even for very large synopsis sizes) and proportional to $1/\epsilon$, where ϵ is the desired approximation guarantee. The key technical idea in our approximation scheme is to make exact DP formulations for probabilistic thresholding *much "sparser"*, while ensuring a maximum relative degradation of ϵ on the quality of the approximate synopsis, i.e., the desired approximation error metric. Extensive experimental results over synthetic and real-life data clearly demonstrate the benefits of our proposed techniques.

1 Introduction

Approximate query processing over compact, precomputed data synopses has attracted a lot of interest recently as a viable solution for dealing with complex queries over massive amounts of data in interactive decision-support and data-exploration environments. For several of these application scenarios, exact answers are not required, and users may in fact prefer fast, approximate answers to their queries. Examples include the initial, exploratory drill-down queries in ad-hoc data mining systems, where the goal is to quickly identify the "interesting" regions of the underlying database; or, aggregation queries in decision-support systems where the full precision of the exact answer is not needed and the first few digits of precision suffice (e.g., the leading digits of a total in the millions or the nearest percentile of a percentage) [1, 2, 5, 11].

Background and Earlier Results. *Haar wavelets* are a mathematical tool for the hierarchical decomposition of functions with several successful applications in signal and image processing [12, 17]. A number of recent studies has also demonstrated the effectiveness of the Haar wavelet decomposition as a data-reduction tool for database problems, including selectivity estimation [13] and approximate query processing over massive relational tables [2, 18] and data streams [8, 14]. Briefly, the key idea is to apply the decomposition process over an input data set along with a thresholding procedure in order to obtain a compact data synopsis comprising of a selected small set of *Haar wavelet coefficients*. The results of the recent research studies of Matias, Vitter and Wang [13, 18], Chakrabarti et al. [2, 3], and others [4, 16] have demonstrated that fast and accurate approximate query processing engines can be designed to operate solely over such compact *wavelet synopses*.

Until very recently, a major criticism of wavelet-based approximate query processing techniques has been the fact that unlike, e.g., random samples, conventional wavelet synopses (such as those used in all the above-cited studies) cannot provide useful guarantees on the quality of approximate answers. The problem here is that coefficients for such conventional synopses are typically chosen in a greedy fashion in order to optimize the overall root-mean-squared (i.e., L_2-norm) error in the data approximation. However, as pointed out by Garofalakis and Gibbons [6, 7], conventional, L_2-optimized wavelet synopses can result in approximate answers of widely-varying quality (even within the same data set) and approximation errors that are heavily biased towards certain regions of the underlying data domain. Their proposed solution, termed *probabilistic wavelet synopses* [6, 7], employs the idea of *randomized coefficient rounding* in conjunction with *Dynamic-Programming-based* thresholding schemes specifically tuned for optimizing the *maximum relative*

error in the approximate reconstruction of individual data values. By optimizing for relative error (with a sanity bound), which is arguably the most important metric for approximate query answers, probabilistic wavelet synopses offer drastic reductions in the approximation error over conventional deterministic techniques and, furthermore, enable unbiased data reconstruction with meaningful, non-trivial *error guarantees* for reconstructed values [6, 7]. While the use of the traditional Haar wavelet decomposition gives the user no knowledge on whether a particular answer is highly-accurate or off by many orders of magnitude, the use of probabilistic wavelet synopses provides the user with an interval where the exact answer is guaranteed to lie into.

Our Contributions. The Dynamic-Programming (DP) algorithms of [6, 7] for constructing probabilistic wavelet synopses are based on an *optimal, continuous DP formulation* over the error-tree structure for Haar coefficients, in conjunction with the idea of *quantizing* the possible choices for synopsis-space allocation using an integer parameter $\mathsf{q} > 1$ (in other words, fractional space is allotted to coefficients in multiples of $1/\mathsf{q}$). Unfortunately, the problem with these exact (modulo the quantization) DP techniques is that they can scale poorly for large data-domain and synopsis sizes – with a domain size of N and synopsis storage of B, the worst-case running time of the optimized algorithm presented in [7] (which uses binary-search to optimize the DP search) is $O(N\mathsf{q}^2 B \log(\mathsf{q}B))$, which becomes $O(N^2\mathsf{q}^2 \log(N\mathsf{q}))$ for large synopsis sizes $B = \Theta(N)$. Our own experience with the DP schemes in [6, 7] has demonstrated that the times required for building a probabilistic wavelet synopsis can increase very rapidly for large domain sizes N and synopsis sizes B; this certainly raises some concerns with respect to the applicability of probabilistic wavelet techniques on massive, real-life data sets. Note that large domain sizes in the range of 10^5–10^7 are not at all uncommon, e.g., for massive time-series data sets where one or more readings/measurements are continuously recorded on every time-tick.

To address these concerns, we propose a novel, fast *approximation scheme* for building probabilistic wavelet synopses over large data sets. Given a quantization parameter q and a desired approximation factor ϵ, our algorithm can be used to build a probabilistic synopsis of *any size* $B \leq N$ in worst-case time of $O(N\mathsf{q} \log N \log \mathsf{q} \log R/\epsilon)$ (where R is roughly proportional to the maximum absolute Haar-coefficient value in the decomposition), while guaranteeing that the quality of the final solution is within a factor of $(1+\epsilon)$ of that obtained by the (exact) techniques of Garofalakis and Gibbons [6, 7] for the same problem instance. In a nutshell, the key technical idea in our proposed approximation scheme is to make the original

DP formulations in [6, 7] *much "sparser"*, while ensuring a maximum relative degradation of $(1+\epsilon)$ on the quality of the approximate solution, i.e., the desired maximum error metric. This is accomplished by restricting the DP search to a carefully-chosen, logarithmically-small subset of *"breakpoints"* that cover the entire range of possible space allotments within the required error guarantee. Our results clearly validate our approach, demonstrating that our algorithm (1) exhibits significantly smaller running times, often by *more than one or even two orders of magnitude*, than the exact DP solution; and, (2) typically produces significantly tighter approximations than the specified $(1+\epsilon)$ factor.

Roadmap. The remainder of this paper is organized as follows. Section 2 gives background material on wavelets, as well as conventional and probabilistic wavelet synopses. In Section 3, we discuss our approximation scheme for constructing probabilistic wavelet synopses in detail. Section 5 describes the results of our empirical study and, finally, Section 6 gives some concluding remarks.

2 Preliminaries

In this section we provide a brief overview of some techniques and algorithms, developed in prior work, that are utilized as helpful tools by our thresholding algorithms.

The Haar Wavelet Transform. Wavelets are a useful mathematical tool for hierarchically decomposing functions in ways that are both efficient and theoretically sound. Broadly speaking, the wavelet decomposition of a function consists of a coarse overall approximation along with detail coefficients that influence the function at various scales [17]. Suppose that we are given the one-dimensional data vector A containing the $N = 8$ data values $A = [2, 2, 0, 2, 3, 5, 4, 4]$. The Haar wavelet transform of A can be computed as follows. We first average the values together pairwise to get a new "lower-resolution" representation of the data with the following average values $[2, 1, 4, 4]$. In other words, the average of the first two values (that is, 2 and 2) is 2, that of the next two values (that is, 0 and 2) is 1, and so on. Obviously, some information has been lost in this averaging process. To be able to restore the original values of the data array, we need to store some *detail coefficients*, that capture the missing information. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average. Thus, in our simple example, for the first pair of averaged values, the detail coefficient is 0 since $2 - 2 = 0$, for the second we again need to store -1 since $1 - 2 = -1$. Note that no information has been lost in this process – it is fairly simple to reconstruct the eight values of the original data array from the lower-resolution array

containing the four averages and the four detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, we get the following full decomposition:

Resolution	Averages	Detail Coefficients
3	[2, 2, 0, 2, 3, 5, 4, 4]	—
2	[2, 1, 4, 4]	[0, -1, -1, 0]
1	[3/2, 4]	[1/2, 0]
0	[11/4]	[-5/4]

The *wavelet transform* (also known as the *wavelet decomposition*) of A is the single coefficient representing the overall average of the data values followed by the detail coefficients in the order of increasing resolution. Thus, the one-dimensional Haar wavelet transform of A is given by $W_A = [11/4, -5/4, 1/2, 0, 0, -1, -1, 0]$. Each entry in W_A is called a *wavelet coefficient*. The main advantage of using W_A instead of the original data vector A is that for vectors containing similar values most of the detail coefficients tend to have very small values. Thus, eliminating such small coefficients from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, resulting in a very effective form of lossy data compression [17]. Furthermore, the Haar wavelet decomposition can also be extended to *multi-dimensional* data arrays through natural generalizations of the one-dimensional decomposition process described above. Multi-dimensional Haar wavelets have been used in a wide variety of applications, including approximate query answering over complex decision-support data sets [2, 18].

Error Tree and Conventional Wavelet Synopses. A helpful tool for exploring the properties of the Haar wavelet decomposition is the *error tree* structure [13]. The error tree is a hierarchical structure built based on the wavelet transform process. Figure 1 depicts the error tree for our example data vector A. Each internal node c_i ($i = 0, \ldots, 7$) is associated with a wavelet coefficient value, and each leaf d_i ($i = 0, \ldots, 7$) is associated with a value in the original data array; in both cases, the index i denotes the positions in the data array or error tree. For example, c_0 corresponds to the overall average of A. The resolution levels l for the coefficients (corresponding to levels in the tree) are also depicted. We use the terms "node" and "coefficient" interchangeably in what follows.

Given a node u in an error tree T, let $\texttt{path}(u)$ denote the set of all proper ancestors of u in T (i.e., the nodes on the path from u to the root of T, including the root but not u) with non-zero coefficients. A key property of the Haar wavelet decomposition is that the reconstruction of any data value d_i depends only

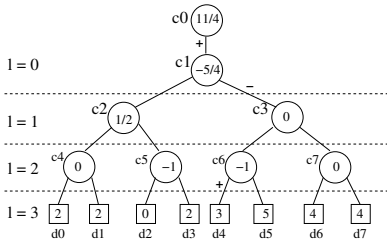

Figure 1: Error tree for our data array A ($N = 8$).

$$M^*[i,\beta_i]=\begin{cases} \min\limits_{\substack{y_i\in(0,\min\{1,\beta_i\}];\\ b_L\in[0,\beta_i-y_i]}} \left\{ \max\left\{ \begin{array}{l} \frac{\mathrm{Var}(i,y_i)}{\mathrm{Norm}(2i)} + M^*[2i, b_L]\,, \\ \frac{\mathrm{Var}(i,y_i)}{\mathrm{Norm}(2i+1)} + M^*[2i+1, \beta_i - y_i - b_L] \end{array} \right\} \right\} & \text{if } i<N,\ c_i\neq 0, \\ & \text{and } \beta_i > 0 \\ \min_{b_L\in[0,\beta_i]} \{\ \max\{\ M^*[2i, b_L]\,,\ M^*[2i+1, \beta_i - b_L]\ \}\ \} & \text{if } i < N \text{ and} \\ & c_i = 0 \\ 0 & \text{if } i \geq N \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

on the values of coefficients on $\mathtt{path}(d_i)$; more specifically, we have $d_i = \sum_{c_j\in\mathtt{path}(d_i)} \delta_{ij} \cdot c_j$, where $\delta_{ij} = +1$ if d_i is in the left child subtree of c_j or $j = 0$, and $\delta_{ij} = -1$ otherwise. For example, in Figure 1, $d_4 = c_0 - c_1 + c_6 = \frac{11}{4} - (-\frac{5}{4}) + (-1) = 3$. The *support region* for a coefficient c_i is defined as the set of (contiguous) data values that c_i is used to reconstruct; the support region for a coefficient c_i is uniquely identified by its index i.

Given a limited amount of storage for building a *wavelet synopsis* of the input data array A, a thresholding procedure retains a certain number $B \ll N$ of the coefficients as a highly-compressed approximate representation of the original data (the remaining coefficients are implicitly set to 0). Conventional coefficient thresholding is a deterministic process that seeks to minimize the overall root-mean-squared error (L_2 *error norm*) of the data approximation [17] by retaining the B largest wavelet coefficients in *absolute normalized value* [17]. L_2 coefficient thresholding has also been the method of choice for the bulk of existing work on Haar-wavelets applications in the data-reduction and approximate query processing domains [2, 13, 14, 18].

Probabilistic Wavelet Synopses. Unfortunately, wavelet synopses optimized for overall L_2 error using the above-described process may not always be the best choice for approximate query processing systems. As observed in a recent study by Garofalakis and Gibbons [6, 7], such conventional wavelet synopses suffer from several important problems, including the introduction of severe bias in the data reconstruction and wide variance in the quality of the data approximation, as well as the lack of non-trivial guarantees for

individual approximate answers. To address these shortcomings, their work introduces *probabilistic wavelet synopses*, a novel approach for constructing data summaries from wavelet-transform arrays. In a nutshell, their key idea is to apply a probabilistic thresholding process based on *randomized rounding* [15], that randomly rounds coefficients either up to a larger *rounding* value or down to zero, so that the value of each coefficient is correct *on expectation*. More formally, each non-zero wavelet coefficient c_i is associated with a *rounding value* λ_i and a corresponding *retention probability* $y_i = \frac{c_i}{\lambda_i}$ such that $0 < y_i \leq 1$, and the value of coefficient c_i in the synopsis becomes a random variable $C_i \in \{0, \lambda_i\}$, where,

$$C_i = \begin{cases} \lambda_i & \text{with probability } \; y_i \\ 0 & \text{with probability } \; 1 - y_i. \end{cases}$$

In other words, a probabilistic wavelet synopsis essentially "rounds" each non-zero wavelet coefficient c_i *independently* to either λ_i or zero by flipping a biased coin with success probability y_i. Note that the above rounding process is *unbiased*; that is, the expected value of each rounded coefficient is $\mathrm{E}[C_i] = \lambda_i \cdot y_i + 0 \cdot (1 - y_i) = c_i$, i.e., the actual coefficient value, while its variance is

$$\mathrm{Var}(i, y_i) = \mathrm{Var}(C_i) = (\lambda_i - c_i) \cdot c_i = \frac{1 - y_i}{y_i} \cdot c_i^2 \tag{2}$$

and the expected size of the synopsis is simply $\mathrm{E}[|\text{synopsis}|] = \sum_{i|c_i \neq 0} y_i = \sum_{i|c_i \neq 0} \frac{c_i}{\lambda_i}$. Thus, since each data value can be reconstructed as a simple linear combination of wavelet coefficients, and by linearity of expectation, it is easy to see that probabilistic wavelet synopses guarantee unbiased approximations of individual data values as well as range-aggregate query answers [6].

Garofalakis and Gibbons [6, 7] propose several different algorithms for building probabilistic wavelet synopses. The key, of course, is to select the coefficient rounding values $\{\lambda_i\}$ such that some desired error metric for the data approximation is minimized while not exceeding a prescribed space limit B for the synopsis (i.e., $\mathrm{E}[|\text{synopsis}|] \leq B$). Their winning strategies are based on formulating appropriate *Dynamic-Programming (DP)* recurrences over the Haar error-tree that explicitly minimize either (a) the maximum normalized standard error (MinRelVar), or (b) the maximum normalized bias (MinRelBias), for each reconstructed value in the data domain. As explained in [6, 7], the rationale for these probabilistic error metrics is that they are directly related to the *maximum relative error* (with an appropriate *sanity bound* S)[1] in the

[1]The role of the sanity bound is to ensure that relative-error numbers are not unduly dominated by small data

approximation of individual data values based on the synopsis; that is, both the MinRelVar and MinRelBias schemes try to (probabilistically) control the quantity $\max_i\{\frac{|\hat{d}_i - d_i|}{\max\{|d_i|, \mathsf{S}\}}\}$, where $\hat{d}_i$ denotes the data value reconstructed based on the wavelet synopsis. Note, of course, that $\hat{d}_i$ is again a *random variable*, defined as the ± 1 summation of all (independent) coefficient random variables on $\mathtt{path}(d_i)$. Bounding the maximum relative error in the approximation also allows for meaningful *error guarantees* to be provided on reconstructed data values [6, 7].

As an example, Equation (1) depicts the DP recurrence in [6, 7] for minimizing the maximum squared Normalized Standard Error (NSE^2) in the data reconstruction, defined as

$$\max_i \text{NSE}^2(\hat{d}_i) \;=\; \max_i \frac{\text{Var}(\hat{d}_i)}{\max\{d_i^2, \mathsf{S}^2\}},$$

where $\text{Var}(\hat{d}_i) = \sum_{c_j \in \mathtt{path}(d_i)} \text{Var}(j, y_j)$. $M^*[i, \beta_i]$ here denotes the minimum maximum value of the *squared* NSE (i.e., NSE^2) among all data values in the subtree of the error-tree rooted at coefficient c_i assuming a space budget of β_i, and $\text{Norm}(i) = \max\{d_{\text{min}}(i)^2, \mathsf{S}^2\}$, where $d_{\text{min}}(i)$ is the minimum absolute data value under c_i's subtree, is a normalization term for that subtree. (Indices $2i$ and $2i+1$ in the recurrence correspond to the left and right child (respectively) of c_i in the error-tree structure (Figure 1).) Intuitively, the DP recurrence in Equation (1) states that, for a given space budget β_i at c_i, the optimal fractional-storage allotments $\{y_i\}$ and the corresponding maximum NSE^2 are fixed by minimizing the larger of the costs for paths via c_i's two child subtrees (including the root in all paths), where the cost for a path via a subtree is the sum of: (1) the variance penalty incurred at c_i itself, assuming a setting of y_i, divided by the normalization term for that subtree, and (2) the optimal cost for the subtree, assuming the given space budget. This minimization, of course, is over all possible values of y_i and, given a setting of y_i, over all possible allotments of the remaining $\beta_i - y_i$ space "units" amongst the two child subtrees of c_i. Of course, if $c_i = 0$ then no space budget needs to be allocated to node i, which results in the simpler recurrence in the second clause of Equation (1). Finally, data-value nodes (characterized by indices $i \geq N$, see Figure 1) cost no space and incur no cost, and the "otherwise" clause handles the case where we have a non-zero coefficient but zero budget ($c_i \neq 0$ and $\beta_i = 0$).

As demonstrated in [6, 7], the DP recurrence in Equation (1) characterizes the optimal solution to the maximum NSE^2 minimization problem for the case of *continuous* fractional-storage allotments $y_i \in (0, 1]$ values [10, 18].

(modulo certain technical conditions that may require small "perturbations" of zero coefficients [6, 7]). A similar DP recurrence can also be given for the maximum normalized bias metric. Their MinRelVar and MinRelBias algorithms then proceed by *quantizing the solution space*; that is, they assume the storage allotment variables y_i and b_L in Equation (1) to take values from a discrete set of choices corresponding to integer multiples of $1/\mathsf{q}$, where $\mathsf{q} > 1$ is an input integer parameter to the algorithms. (Larger values of q imply results closer to the optimal, continuous solution.) Furthermore, both MinRelVar and MinRelBias cap the variance of a coefficient c at c^2, thus allowing for zero-space allotments to unimportant coefficients (this also implies that non-zero allotments of size $\leq \frac{1}{2}$ are useless, as they result in larger variance (Equation (2)) while utilizing more space).The running time of their (quantized) MinRelVar and MinRelBias algorithms is $O(N\mathsf{q}^2 B \log(\mathsf{q}B))$ with an overall space requirement of $O(N\mathsf{q}B)$ (and an in-memory working-set size of $O(\mathsf{q}B \log N)$); furthermore, their techniques also naturally extend to multi-dimensional data and wavelets, with a reasonable increase in time and space complexity [7]. Experimental results over synthetic and real-life data in [6, 7] have demonstrated the superiority of MinRelVar and MinRelBias probabilistic synopses as an approximate query answering tool over conventional wavelet synopses. In our discussion, we use $M^*_{\mathsf{q}}[i, \beta_i]$ to denote the result of the quantized (exact) algorithms of [6, 7] (e.g., maximum NSE^2 for MinRelVar) for the error subtree rooted at coefficient c_i assuming a space budget of β_i.

3 Our Approximation Scheme

In this section, we present our efficient approximation scheme, termed ϵ-ApproxRV, for constructing probabilistic wavelet synopses over large data sets. Our ϵ-ApproxRV is a guaranteed $(1 + \epsilon)$ approximation algorithm for the MinRelVar scheme of Garofalakis and Gibbons [6, 7]; that is, it focuses on minimizing the maximum NSE^2 in the data reconstruction. Our techniques can easily be extended to handle other error metrics, such as the maximum normalized bias employed by MinRelBias [6, 7]. We here present our ϵ-ApproxRV algorithm for the case of one-dimensional Haar wavelets, and defer the extensions to multiple dimensions for the full paper.

3.1 The One-Dimensional ϵ-ApproxRV Algorithm

Consider the error-tree structure for a one-dimensional Haar wavelet decomposition, and let R denote the *maximum absolute normalized value* of any coefficient in the tree, defined as

$$R \;=\; \max_i \frac{|c_i|}{\max\{d_{\min}(i), \mathsf{S}\}},$$

where, as previously, $d_{\min}(i)$ denotes the minimum absolute data value in the subtree of node i. (Typically, e.g., for frequency-count vectors, the denominator in the above expression is > 1, which implies that R is in the order of the maximum absolute coefficient value.) Our ϵ-ApproxRV algorithm runs in $O(\frac{N\mathsf{q}\log N\log \mathsf{q}\log R}{\epsilon})$ time and computes an approximate solution for all possible values of the synopsis space budget $B \leq N$; the corresponding time complexity of the exact MinRelVar algorithm (for $B = \Theta(N)$) is significantly higher: $O(N^2\mathsf{q}^2\log(N\mathsf{q}))$ [6, 7]. Again, the key idea in our ϵ-ApproxRV algorithm is to speed up the DP search by making it much "sparser"[2] – in a nutshell, our approximate "sparse" DP algorithm will only search over a few possible space allotments for each error subtree, which are carefully chosen to guarantee a maximum deviation of $(1+\epsilon)$ from the optimal solution. Our ϵ-ApproxRV algorithm proceeds in a bottom-up fashion over the input error tree – to simplify the exposition in this section, we assume that levels in the error tree are numbered bottom-up, with leaf-node coefficients at level 0 and the root (overall average) at level $\log N - 1$.

Fix a quantization parameter q, and let $M_{\mathsf{q}}[v, \beta_v]$ denote the approximate maximum squared NSE (NSE^2) computed by ϵ-ApproxRV for any data value in the error subtree rooted at node v. As earlier, $M^*_{\mathsf{q}}[v, \beta_v]$ is the corresponding optimal NSE^2 value computed by MinRelVar. Note that, for any node v, the $M^*_{\mathsf{q}}[v, \beta_v]$ values are clearly *monotonically decreasing* in β_v; that is, $M^*_{\mathsf{q}}[v, x] \leq M^*_{\mathsf{q}}[v, y]$ for $x > y$ [7].

For the base case, consider a leaf-node coefficient v (at level 0) – clearly, in this case

$$M^*_{\mathsf{q}}[v, \beta_v] = \frac{\text{Var}(c_v, \min\{1, \beta_v\})}{\min\{\text{Norm}(2v), \text{Norm}(2v+1)\}}$$

i.e., the maximum normalized variance of the corresponding random variable with a success probability of β_v (values of β_v larger than 1 obviously result in zero normalized variance). It is easy to see that all possible values for $M^*_{\mathsf{q}}[v, \beta_v]$, for any β_v value, can be computed in time $O(\mathsf{q})$, where q is the designated quantization parameter. Out of these $O(\mathsf{q})$ variance values and possible allotments to c_v, our ϵ-ApproxRV algorithm picks a subset of allotments $b_1 > \ldots > b_h$ such that: (1) for each allotment $x \in [b_i, b_{i-1})$ we have $M^*_{\mathsf{q}}[v, b_i] \leq (1+\epsilon)M^*_{\mathsf{q}}[v, x]$; and, (2) b_1 through b_h cover the entire possible range of space allotments to c_v, i.e., $b_1 = 1$ and $b_h = 0$. This can obviously be done in $O(\mathsf{q})$ time by simply going over all M^*_{q} values

[2] Guha et al. [9] also discuss sparse DP algorithms in an entirely different context, namely in building approximate V-optimal histograms over data streams.

and selecting b_{i+1} as the first allotment $\leq b_i$ such that $M^*_{\mathsf{q}}[v, b_{i+1}] > (1+\epsilon)M^*_{\mathsf{q}}[v, b_i]$. Since the maximum normalized variance for a coefficient value c_v is at most (see Section 2) $\frac{c_v^2}{\min\{\text{Norm}(2v),\text{Norm}(2v+1)\}} \leq R^2$, is easy to see that the number h of allotment "breakpoints" selected in this fashion is at most $O(\log_{1+\epsilon} R^2) = O(\frac{\log R}{\log(1+\epsilon)}) \approx O(\frac{\log R}{\epsilon})$ (for small values of $\epsilon < 1$). The approximate error values determined by our ϵ-ApproxRV algorithm for coefficient c_v are defined only by these h breakpoints $b_1, \ldots, b_h$ – specifically, $M_{\mathsf{q}}[v, b_i] = M^*_{\mathsf{q}}[v, b_i] = \frac{\text{Var}(c_v, b_i)}{\min\{\text{Norm}(2v),\text{Norm}(2v+1)\}}$ for $i = 1, \ldots, h$, and for any other possible allotment $x \in [b_i, b_{i-1})$, we define $M_{\mathsf{q}}[v, x] = M_{\mathsf{q}}[v, b_i]$. Thus, it is easy to see that, by construction, the approximation error of our ϵ-ApproxRV algorithm is bounded by a factor of $(1+\epsilon)$ at leaf coefficients (at level 0); in other words, all dropped allotments are "covered" by a logarithmic number of breakpoints to within an (1 + ϵ) factor.

Now, proceeding inductively, consider an internal error-tree node v at level j, with children u and w (at level $j-1$), and assume that the subtree rooted at u (w) has determined a collection of l_u (resp., l_w) error-function breakpoints $a_1 > \ldots > a_{l_u}$ (resp., $b_1 > \ldots > b_{l_w}$), and corresponding approximate NSE^2 values $M_{\mathsf{q}}[]$, that cover the range of allotments to each subtree and such that, for each $x \in [a_i, a_{i-1})$ ($i = 2, \ldots, l_u$), we have $M_{\mathsf{q}}[u, a_i] \leq (1+\epsilon)^j M^*_{\mathsf{q}}[u, x]$ (and similarly for w). Our ϵ-ApproxRV algorithm computes the allotment breakpoints and approximate error values $M_{\mathsf{q}}[]$ at the parent node v by iterating over all possible space allotments to node v and the breakpoints determined by the u and w subtrees (rather than all possible allotments to child subtrees), and retaining the minimum M_{q} values for each total allotment. The following lemma shows that, for each fixed space allotment to the coefficient at node v, it actually suffices to look at only $l_u + l_w$ combinations (a_i, b_k) for the subtree allotments rather than all possible $l_u l_w$ combinations.

Lemma 1: *When minimizing the maximum (approximate)* NSE^2 *error at node v, for any fixed space allotment to node v, it suffices to consider only $l_u + l_w$ combinations of allotments (a_i, b_k) to the child subtrees rooted at u, w.* ∎

Proof: Assume a fixed space allotment to the coefficient at node v, and let $leftVar$ ($rightVar$) denote the variance of node v (for the given allotment) divided by the normalization factor of its left (resp., right) subtree. Let L_u denote the sorted list of approximate NSE^2 values $M'_{\mathsf{q}}[u, a_i] = M_{\mathsf{q}}[u, a_i] + leftVar$, i.e., $M_{\mathsf{q}}[u, a_1] + leftVar < \ldots < M_{\mathsf{q}}[u, a_{l_u}] + leftVar$, with L_w defined similarly using the $rightVar$ quantity and the $M_{\mathsf{q}}[w, b_k]$ entries. Let $L = \text{merge}(L_u, L_w)$, i.e., $M'_{\mathsf{q}}[y_1] \leq \ldots \leq M'_{\mathsf{q}}[y_{l_u+l_w}]$, where

$y_i \in \{(u, a_k) : k = 1, \ldots, l_u\} \cup \{(w, b_k) : k = 1, \ldots, l_w\}$. Now assume that a_i space is allocated to the u-subtree of v. Then, it is easy to see that, when considering the allotment to the w-subtree, out of all the b-values that lie to the left of a_i in L we really only need to consider the rightmost b-value, say b_k – the reason of course is that lower values of $M'_{\mathsf{q}}[w, b]$ (i.e., allotments $b > b_k$) result in configurations that use more total space without improving the error at v (since that is dominated by the u-subtree). These configurations are clearly useless in our error-minimization procedure. For the b-values to the right of a_i in L, a similar argument again applies: when a value b_k is assumed, only the closest a-value to its left in L needs to be considered. ∎

Thus, our approximate error-minimization procedure at v only needs to consider, for each fixed space allotment $s = 0, 1/\mathsf{q}, \ldots, 1$ to node v, $l_u + l_w$ breakpoint combinations (a_i, b_k) for the u and w subtrees, which can be determined easily in $O(l_u + l_w)$ time based on the proof of Lemma 1. Let $\mathcal{S}(s)$ denote the list of the obtained $l_u + l_w$ (a_i, b_k) combinations for each space allotment s to node v. The sorted list of approximate error values at node v can be computed in $O(\mathsf{q}(l_u+l_w) \log \mathsf{q})$ time by merging these lists using a heap structure or, alternatively, pair-wise merging them in $\log \mathsf{q}$ steps. Thus, an initial list of $O(\mathsf{q}(l_u+l_w))$ breakpoints for the v subtree is determined based on the "useful" space-allocation configurations found through the above equation – clearly, configurations that give the same (or, larger) NSE^2 values for the same (or, larger) amount of total space are useless and should be discarded. In other words, we define the initial set of space-allotment breakpoints for the v subtree as $\mathcal{C} = \{c = a_i+b_k+s : M_{\mathsf{q}}[v, a_i+b_k+s] \leq M_{\mathsf{q}}[v, a+b+t]$ for all $a + b + t \leq a_i + b_k + s\}$. (Useless configurations and configurations with space larger than B can easily be discarded in the merging pass for the $O(\mathsf{q})$ sub-lists described above.) It is easy to verify that, based on our inductive assumption, this set of breakpoints $\mathcal{C}$ covers the entire range of possible allotments for the v subtree; furthermore, the following lemma shows that it also preserves the approximation properties guaranteed by the individual subtrees.

Lemma 2: *Let $s_1 > \ldots > s_h$ denote the sorted list of space-allotment breakpoints $\mathcal{C}$ for the v subtree, computed as described above, and let $x \in [s_i, s_{i-1})$ for any i. Then, $M_{\mathsf{q}}[v, s_i] \leq (1 + \epsilon)^j M^*_{\mathsf{q}}[v, x]$, where j is the level of node v.* ∎

Proof: Assume that the optimal error value $M^*_{\mathsf{q}}[v, x]$ is obtained through the allotment configuration (y_u, y_w, s), that is:

$$M^*_{\mathsf{q}}[v,x] = \max \left\{ \begin{array}{l} \frac{Var(c_v,s)}{\mathrm{Norm}(u)} + M^*_{\mathsf{q}}[u, y_u] \\ \frac{Var(c_v,s)}{\mathrm{Norm}(w)} + M^*_{\mathsf{q}}[w, y_w] \end{array} \right.$$

where, of course, $x \geq y_u + y_w + s$. Since the breakpoints for the u and w subtrees cover all possible allotments, let $y_u \in [a_i, a_{i-1})$ and $y_w \in [b_k, b_{k-1})$. By our inductive hypothesis, it is easy to see that the configuration (a_i, b_k, s) (which is obviously examined by the ϵ-ApproxRV algorithm) will give $M_{\mathsf{q}}[v, a_i + b_k + s] \leq (1+\epsilon)^j M^*_{\mathsf{q}}[v,x]$ and, clearly, $a_i + b_k + s \leq s_i \leq x$. Since $M_{\mathsf{q}}[v, s_i] \leq M_{\mathsf{q}}[v, a_i + b_k + s]$, the result follows. ∎

A potential problem with our approximation scheme, as described so far, is that the list of space-allotment breakpoints $\mathcal{C}$ would appear to grow exponentially as the DP moves up the error-tree levels. (So, starting with r breakpoints at the leaf nodes, we get $O(\mathsf{q}^j 2^j r)$ breakpoints at level j of the tree.) However, not all s_i's in $\mathcal{C}$ are necessary – we can actually "trim" $\mathcal{C}$ to a small number of breakpoints, while incurring an additional $(1+\epsilon)$ worst-case factor degradation on our approximation error. We perform this trimming process at every node of the error tree (except for the final, root node). More specifically, assume a chain of computed breakpoints $s_{i-k} > s_{i-k+1} > \ldots > s_i$ such that, for each $l = i-k, \ldots, i-1$ we have $M_{\mathsf{q}}[v, s_i] \leq (1+\epsilon) M_{\mathsf{q}}[v, s_l]$. Then, clearly, $M_{\mathsf{q}}[v, s_i]$ can "cover" all the points that are covered by $s_{i-k}, \ldots, s_{i-1}$ at an additional $(1+\epsilon)$ degradation, since, for any $l = i-k, \ldots, i-1$:

$$M_{\mathsf{q}}[v, s_i] \leq (1+\epsilon) M_{\mathsf{q}}[v, s_l] \leq (1+\epsilon)^{j+1} M^*_{\mathsf{q}}[v,x] \qquad \forall x \in [s_l, s_{l-1}).$$

Thus, in this situation, the allotment points $s_{i-k}, \ldots, s_{i-1}$ can be eliminated and s_i can cover their ranges to within a $(1+\epsilon)^{j+1}$ factor. Now, note that the maximum value of the overall NSE^2 value at level j (and, thus, the range of values for the $M_{\mathsf{q}}[]$ array) is certainly upper-bounded by $(j+1)R^2$. This means that the total number of breakpoints obtained in the manner described above is at most $\log_{1+\epsilon}((j+1)R^2) \approx O(\frac{\log(j+1)+\log R}{\epsilon})$, which is an upper bound for the size of our "breakpoint" list constructed at level j of the error tree. Thus, with an overall computational effort of

$$\begin{aligned} \sum_{j=0}^{\log N - 1} O(\frac{N}{2^{j+1}} \frac{\mathsf{q} \log \mathsf{q} (\log(j+1) + \log R)}{\epsilon}) &\leq O(\frac{N \mathsf{q} \log \mathsf{q} \log R}{\epsilon} \sum_{j=0}^{\log N - 1} \frac{1}{2^{j+1}}) + \\ + O(\frac{N \mathsf{q} \log \mathsf{q}}{\epsilon} \sum_{j=0}^{\log N - 1} \frac{j+1}{2^{j+1}}) &= O(\frac{N \mathsf{q} \log \mathsf{q} \log R}{\epsilon}), \end{aligned}$$

we get a (collection of) approximate solutions at the root node of the error tree that are guaranteed to cover the optimal MinRelVar solutions to within a $(1+\epsilon)^{\log N}$ factor. Then, it is easy to see that, setting $\epsilon' = \epsilon/\log N$, we get a guaranteed $(1+\epsilon)$ approximation in time $O(\frac{N\mathsf{q}\log\mathsf{q}\log N\log R}{\epsilon})$. Note that, to find the approximate solution for any specific choice of the allotment space B, we simply start out at the root and re-trace the steps of the algorithm for the largest root breakpoint s_i that is $\leq B$; to do that, we just need to keep track of the (a_i, b_k, s) configuration that generated each of the breakpoints at each error tree node and proceed recursively down the tree. It is easy to see that the overall space required by the ϵ-ApproxRV algorithm is

$$\sum_{j=0}^{\log N-1} O(\frac{N}{2^{j+1}}\frac{\log(j+1)+\log R}{\epsilon'}) = O(\frac{N\log N\log R}{\epsilon}),$$

while the working set size (maximum amount of memory-resident data) is only $O(\frac{\mathsf{q}(\log\log N+\log R)}{\epsilon'}) = O(\frac{\mathsf{q}\log N(\log\log N+\log R)}{\epsilon})$. To see this, note that ϵ-ApproxRV works in a bottom-up fashion and, when computing the breakpoint-list for a given node v, we only need access to (1) the $l_u + l_w$ breakpoints of its child nodes in the error tree; and, (2) the $O(\mathsf{q}(l_u+l_w))$ initial breakpoints for node v that are computed just before the trimming process. Thus, the maximum working set will occur in the top-level of the error tree (level $\log N$-1), where $l_u + l_w = O(\frac{\log R+\log\log N}{\epsilon'})$. Finally, note that the overall space required by our ϵ-ApproxRV algorithm can never exceed the space requirements of MinRelVar (i.e., $O(N\mathsf{q}B)$), since for each of the N nodes in the error tree we cannot have more than $\mathsf{q}B$ different breakpoints. The following theorem summarizes the results of our analysis for the one-dimensional ϵ-ApproxRV algorithm.

Theorem 3: *The ϵ-ApproxRV algorithm correctly computes a list of breakpoints at the root node such that, for any space budget $B \leq N$ and approximation factor ϵ, the estimated maximum NSE^2 value is within a factor of $(1+\epsilon)$ from the optimal MinRelVar solution. The overall ϵ-ApproxRV computation requires $O(\frac{N\mathsf{q}\log\mathsf{q}\log N\log R}{\epsilon})$ time and $O(\frac{N\log N\log R}{\epsilon})$ space, with a working set size of $O(\frac{\mathsf{q}\log N(\log\log N+\log R)}{\epsilon})$.*

∎

3.2 Analyzing the Complexity of ϵ-ApproxRV

Both the running time and space complexities of the ϵ-ApproxRV algorithm represent a significant improvement over the MinRelVar algorithm when $\log N = O(B)$. Moreover, as we describe in Section 3.3, the ϵ-ApproxRV algorithm can be slightly modified to calculate all the necessary breakpoints in one pass, thus making it suitable for large data sets that do not fit in main memory. On the contrary, the MinRelVar algo-

rithm does not exhibit such a desirable characteristic, and its performance may deteriorate significantly in such data sets.[3]

A natural question that arises is whether the ϵ-ApproxRV algorithm will get outperformed for small synopsy sizes ($B = o(\log N)$). When considering the space requirements of the ϵ-ApproxRV algorithm, one observes that the number of distinct breakpoints at each node of the error tree cannot possibly exceed the value $\mathsf{q} \times B$. The ϵ-ApproxRV algorithm, thus, cannot require assymptotically more space than the MinRelVar algorithm, independently of the values of B and ϵ. Similarly, the list of breakpoints at each node can be calculated in a running time of: $O(\mathsf{q} \times (\mathsf{q}B) \log \mathsf{q}) = O(B\mathsf{q}^2 \log \mathsf{q})$. Thus, the overall running time complexity is bounded by $O(NB\mathsf{q}^2 \log \mathsf{q})$, an improvement over MinRelVar, even for very small values of B.

An important observation that we need to make is that the running time and space complexities of the ϵ-ApproxRV algorithm are based on a worst-case analysis and scenarios that are not very likely to occur in real-life data sets. For example, one of the advantages of the wavelet decomposition process is that in data sets with similar data values, several of the produced wavelet coefficients have either zero, or near-zero values and, thus, not storing them only introduces small errors in the approximation. A similar observation is also true when trying to minimize the maximum relative error of the approximation, meaning that many of the produced coefficients will have an absolute normalized value much smaller than R. This, in turn, implies that in many nodes of the error tree the maximum value of the overall NSE^2 value at level j will be significantly smaller than $(j+1)R^2$. This results not only in reduced space requirements by the ϵ-ApproxRV algorithm, since smaller lists of breakpoints are stored in each node than in the worst-case analysis, but also to a reduced running time, since smaller lists of breakpoints are scanned and merged. Moreover, notice that any breakpoints that correspond to space larger than B are automatically trimmed by our approach, something not depicted in the worst-case complexity analysis of the ϵ-ApproxRV algorithm. For the above reasons, and as our experimental evaluation will demonstrate, the ϵ-ApproxRV algorithm manages to significantly outperform the MinRelVar algorithm even for small values of B and ϵ.

[3] A modified version of MinRelVar can also be applied over a single pass, but exhibits an increased running time complexity of $O(N\mathsf{q}^3B^2)$.

3.3 Improving I/O Complexity

While the ϵ-ApproxRV algorithm that we have detailed so far contains all the essential concepts for an efficient in-memory computation of the necessary breakpoints in all nodes, it may still be, as described above, inefficient when applied to very large data sets. The algorithm's main drawback is its space requirements, which are asymptotically larger than the size of the data set. In large domain sizes, the produced breakpoints will not fit in main memory and the I/O incurred during the algorithm's execution may be substantial.

A simple modification to resolve this issue is to calculate the breakpoints of the nodes in the error tree using a postorder traversal. A sketch of the I/O efficient ϵ-EfficientApproxRV algorithm is presented in Figure 2.[4] The modified ϵ-EfficientApproxRV algorithm operates on the nodes in the same manner in which the wavelet coefficients are calculated during the decomposition process (Line 7). The algorithm requires as input the index v of the currently processed coefficient in the error tree, the space constraint B, the quantization parameter q, the domain size N and the ϵ factor. The return value of the ϵ-EfficientApproxRV algorithm is the average data value within the node's support region (Section 2), and is used in the calculation of the wavelet coefficients. Lines 1-4 simply serve as boundary cases. For any node v in the error tree, the algorithm first performs the joint decomposition process and calculation of the breakpoints for its left and right subtrees (Lines 5-6) and then calculates the coefficient value c_v (Line 7). Then, for each meaningful space allotment (which must be a multiple of $1/\mathsf{q}$) to the node's coefficient (Lines 9-11), it calculates a list of at most $|L_{2\times v}| + |L_{2\times v+1}|$ breakpoints (where $L_{2\times v}$ and $L_{2\times v+1}$ are the breakpoint lists of the node's left and right subtrees, correspondingly). These lists are then merged in one pass, in which a trimming process (described in the Section 3.1) that incurs an additional (1+ϵ) degradation at each level is also performed (Lines 12-13).

Memory Requirements. In order for the modified ϵ-EfficientApproxRV algorithm to be I/O efficient, for each processed node v we need to keep into main memory the breakpoint lists calculated at its children nodes. Due to the recursive nature of the algorithm, node v will be processed immediately after its right child. At any point during the algorithm's execution, only nodes belonging in the path from v to the root of the error tree need to keep the breakpoint lists of their children nodes in main memory. All other nodes have either already computed their breakpoint lists, or have not been visited by the algorithm. Since at most $O(logN)$ nodes may exist in any path from a node to the root of the error tree, the algorithm can compute

[4] The presented algorithm can be applied to any node containing two subtrees. The algorithm's extension for the root of the error tree is straightforward.

procedure ϵ-EfficientApproxRV(v, B, q, N, ϵ)
Input: index/coordinate v of current coefficient; space constraint B;
quantization parameter $\mathsf{q} > 1$; domain size N;
Output: list L_v of breakpoints; returns average of data values in node's subtree

1. **if** $v \geq N$ **then**
2. $L_v = 0$
3. return 0
4. **endif**
   // traverse children nodes first
5. leftAvg = ϵ-EfficientApproxRV($2 \times v$, B, q, S)
6. rightAvg = ϵ-EfficientApproxRV($2 \times v + 1$, B, q, S)
7. $c_v = \frac{leftAvg - rightAvg}{2}$ *// calculate value of coefficient*
8. **if** $c_v > 0$ **then** $maxAllot = \mathsf{q}$ **else** $maxAllot = 0$
9. **for** $s := 0$ **to** $maxAllot$ **do**
10. Calculate breakpoint list $PartialL_j$ for space allotment s/q to c_v
    // at most $|L_{2v}| + |L_{2v+1}|$ breakpoints if v is non-leaf
11. **endfor**
12. Merge all $maxAllot + 1$ $PartialL$ lists and trim resulting list based on B, ϵ
13. Store resulting breakpoint list in L_v
14. return (leftAvg + rightAvg)/2

Figure 2: The I/O Efficient ϵ-EfficientApproxRV Algorithm

all breakpoints in one pass of the data with a memory size of:

$$\sum_{j=0}^{\log N - 1} O(\frac{\log(j+1) + \log R}{\epsilon'}) = O(\frac{(\log N)^2(\log R + \log\log N)}{\epsilon}),$$

In this case, the algorithm will incur a total I/O cost (for a disk page size of P) of: $O(\frac{N}{P} + \frac{N \log N \log R}{P \times \epsilon}) = O(\frac{N(\log N \log R + \epsilon)}{P \times \epsilon})$.

4 Extensions to Multi-Dimensional Wavelets

4.1 Key Ideas

We now discuss the key ideas for extending our techniques to multi-dimensional data sets. (For a detailed discussion of multi-dimensional wavelets we refer the reader to the analysis presented in [7].) Briefly, for a D-dimensional data set, the error-tree structure becomes significantly more complex. Each node in the error tree (besides the root node) corresponds to a *set* of (at most) $2^D - 1$ wavelet coefficients with the same support region, but different signed contributions for each region quadrant. Furthermore, each error-tree node i (besides the root) may have up to 2^D children, corresponding to the quadrants of the common

support region for all coefficients in i.

Extension of MinRelVar to Multiple Dimensions. The dynamic programming solution of [6] considers at each node of the error tree the optimal way of distributing the available space to its child subtrees, and to the coefficient values of the node itself. We now explain how this is achieved by appropriately altering the DP formulation of Equation (1). A more detailed description of the approach can be found in [7]. In Equation (1), for the 1-dimensional case, we considered allocating space B to a subtree rooted at the node with index i. The multi-dimensional formulation considers allocating space B to a list of subtrees rooted at nodes $\langle i_1, i_2, \ldots, i_k \rangle$. To achieve this, every time this input list contains more than one subtrees, we consider the optimal assignment of space b_L to the first subtree of the list (rooted at node i_1), and assigning the remaining $B - b_L$ space to the list of the remaining subtrees $\langle i_2, \ldots, i_k \rangle$. If the input list contains just a single subtree, then the algorithm considers the optimal allocation of space y_j to the root node j of the subtree (which in this case contains at most $2^D - 1$ coefficients), of space b_L to the first child subtree of node j, and of space $B - y_j - b_L$ to the list of remaining child subtrees. The space partitioning process in the root node of the subtree among its coefficients can be computed optimally with a computation cost of $O(D2^D)$, as shown in [7]. Finally, similarly to the 1-dimensional case, the algorithms in [6, 7] quantize the search space and consider space allotments that are a multiple of 1/q to each list of subtrees.

Extending the ϵ-ApproxRV Algorithm to Multiple Dimensions. We now show how to extend the ϵ-ApproxRV algorithm to multi-dimensional data sets. Similarly to the MinRelVar algorithm, we consider allocating a space budget B to a list L of subtrees, and carefully choose a set of space allotments that will guarantee a maximum deviation of ϵ from the optimal DP solution. To achieve this, we modify the ϵ-ApproxRV algorithm to handle the two cases mentioned above: (1) When the input list L contains more than one subtrees; and (2) When the input list L contains a single subtree.

• **Handling Input Lists with Multiple Subtrees.** When the input list L consists of more than one subtrees, we need to decide how much space to assign to the first subtree of the list, and then assign the remaining space budget to the list $Tail$ of the remaining subtrees. This case is similar to the process that we followed in the 1-dimensional case in order to determine for each node and for each fixed space allotment to the node's coefficient, the set of $l_u + l_w$ space allotments that we needed to consider. Here, we again have to consider combining space allotments from two lists; the first list corresponds to the space allotments of the first subtree, and the second list is the space allotments of the list $Tail$ of the remaining subtrees.

However, in this procedure we no longer have to take into account, in this step, the normalized variance of any coefficient value. The merging procedure is, therefore, identical to the one described in Section 3, if we set $leftVar = rightVar = 0$. Similarly to the analysis in Section 3, if the root nodes of the subtrees belong to level j and the maximum level of the error tree is H, then the number of l_u space allotments will be $O(\frac{\log(j+1)+\log R}{\epsilon'})$, where $\epsilon' = \epsilon/H$. Therefore, for any list of K subtrees with root nodes at level j, the number of resulting space allotments will be $O(\frac{K(\log(j+1)+\log R)}{\epsilon'})$. Finally, to determine the normalization term for the list L of subtrees, we consider the minimum absolute data value $d_{\min}(L)$ under any of the subtrees in L, and set $\text{Norm}(L) = \max\{d_{\min}(L)^2, \mathsf{S}^2\}$.

• **Handling Input Lists with a Single Subtree.** If the input list L contains just a single subtree, then in the ϵ-ApproxRV algorithm we need to determine a list of space allotments in the root node v of this subtree, that guarantees a maximum deviation of ϵ from the optimal solution. For any fixed space allotment to v, we can use the algorithm in [7] to optimally partition this space to the node's coefficient values and calculate v's variance as the sum of variances of its coefficient values. Let L_u (L_w) denote the list of space allotments of v's first (resp., list of $K-1$ remaining) subtree(s). Then, with an identical procedure to the one of Section 3, we can select, for each fixed space allotment to v, a list of $|L_u| + |L_w|$ sorted space allotments for node v. The only differences are: (1) The number of different space allotments to node v is $O(\mathsf{q}2^D)$; and, (2) The merging of these lists, therefore, requires time $O(\mathsf{q}2^D \log(\mathsf{q}2^D)(|L_u| + |L_w|)) = O(\frac{\mathsf{q}2^D(\log \mathsf{q}+D)K(\log(j+1)+\log R)}{\epsilon'})$, where j is the level of v.

• **Running Time and Space Complexities.** For each node in the error tree, there are at most $O(2^D)$ different lists of its children subtrees (assuming a fixed ordering of these subtrees). For a node v at level j, in order to compute its space allotments, we need to first compute the space allotments in all $O(2^D)$ lists containing subtrees rooted at children nodes of v. This procedure dominates the computation time in the node and requires a computational effort of: $O(\mathsf{q}2^D(\log \mathsf{q} + D)\sum_{i=1}^{2^D} \frac{i(\log(j+1)+\log R)}{\epsilon'}) = O(\frac{\mathsf{q}(\log \mathsf{q}+D)8^D(\log(j+1)+\log R)}{\epsilon'})$. Let N denote the total number of cells in the multi-dimensional data array and $maxD$ denote the maximum domain size of any dimension. Then, the overall running time is $O(\frac{N\mathsf{q}\log(\mathsf{q}+D)\log maxD \log R 8^D}{\epsilon})$, while the space requirements are $O(\frac{N \log maxD \log R 4^D}{\epsilon})$. Note, of course, that in most real-life scenarios employing wavelet-based data reduction, the number of dimensions is typically a small constant (e.g., 2–6) [2, 5, 7].

4.2 Improving the Running Time

In the wavelet decomposition process of a multi-dimensional data set, the number of non-zero coefficients produced may be significantly larger than the number N_z of non-zero data values. In [7] the authors proposed an adaptive coefficient thresholding procedure that retains at most N_z wavelet coefficients *without* introducing any reconstruction bias. Using this procedure, the authors in [7] demonstrated how the MinRelVar algorithm can be modified so that its running time and space complexities have a dependency on N_z, and not on N (i.e., the total number of cells in the multi-dimensional data array). It would thus be desirable if the ϵ-ApproxRV algorithm could be modified in a similar way, in order to decrease its running time and space requirements.

Let N_z denote the number of error tree nodes that contain non-zero coefficient values, possibly after the aforementioned thresholding process. We will first illustrate that for any node in the error tree containing zero coefficient values, and which has at most one node in its subtree that does contain non-zero coefficient values, no computation is needed. Equivalently, our algorithm will need to compute breakpoint lists in only: (i) nodes containing non-zero coefficient values; or (b) nodes that contain zero coefficient values, but which are the least common ancestor of at least two non-zero tree nodes beneath it in the error tree.

Let k be a node that is the only node in its subtree with non-zero coefficient values. Obviously we do not need to consider the breakpoint lists in the descendant nodes of k, since the NSE^2 values in them will be zero. We can thus calculate the breakpoint list in k by treating it as a leaf-node. An important observation is that for any ancestor of k that contains just a single non-zero error tree beneath it (which is certainly the subtree of node k), no computation is necessary, since the breakpoint lists of k can always be used instead. The only additional computation is needed in any node n with zero-coefficients that has at least two non-zero error tree nodes beneath it in the error tree (in different subtrees). In this case, the breakpoint list of node n needs to be calculated, using as input the breakpoint lists of its non-zero descendant tree nodes. It is easy to demonstrate that at most $N_z - 1$ such nodes may exist. Thus, the ϵ-ApproxRV algorithm will need to calculate the breakpoint lists in at most $O(2N_z - 1){=}O(N_z)$ nodes, thus yielding running time and space complexities of $O(\frac{N_z \mathsf{q} \log(\mathsf{q}+D) \log maxD \log R 8^D}{\epsilon})$ and $O(\frac{N_z \log maxD \log R 4^D}{\epsilon})$, respectively. We here need to note that in order to implement our algorithm as described here, we need to sort the N_z coefficients based on their postorder numbering in the error tree.[5] This requires an additional $O(N_z \log N_z)$ time for the sorting

[5]Notice that this ordering is similar to the one suggested in Section 3.3 for an I/O efficient computation of the breakpoints.

process. However, this running time is offen significantly smaller than the benefits of having running time and space dependencies based on N_z, rather than on N.

5 Experimental Study

In this section, we present an extensive experimental study of our proposed ϵ-ApproxRV algorithm for constructing probabilistic wavelet synopses over large data sets. The objective of this study is to evaluate both the scalability and the obtained accuracy of our proposed ϵ-ApproxRV algorithm when compared to the dynamic programming algorithm MinRelVar of [6, 7] for a large variety of real-life and synthetic data sets. For the later DP solution, we used the significantly faster version of the algorithm that was very recently proposed in [7]. The main findings of our study for the ϵ-ApproxRV algorithm include:

• **Near Optimal Results.** The ϵ-ApproxRV algorithm consistently provides near-optimal solutions. Moreover, the actual deviation of the ϵ-ApproxRV solution from the optimal one is typically significantly smaller (usually by a factor larger than 5) than the specified ϵ value.

• **Significantly Faster Solution.** Our ϵ-ApproxRV algorithm provides a fast and scalable solution for constructing probabilistic synopses over large data sets. Compared to the MinRelVar algorithm of [7], the running time of the ϵ-ApproxRV algorithm is often more than an order of magnitude (and in some times more than two orders of magnitude) smaller, while at the same time providing highly-accurate answers. In fact, the ϵ-ApproxRV algorithm is significantly faster even for cases when the optimal solution is required (ϵ = 0).

5.1 Testbed and Methodology

Techniques and Parameter Settings. Our experimental study compares the ϵ-ApproxRV and MinRelVar algorithms for constructing probabilistic wavelet synopses. Both algorithms utilize the quantization parameter q, which is assigned a value of 10, as suggested in [6], in our experiments. Larger values of this quantization parameter improved the running time performance of the ϵ-ApproxRV algorithm when compared to the MinRelVar algorithm, as expected by the running time complexities of the two algorithms. Finally, the sanity bound of each data set is set to its 5%-quantile data value.

Data Sets. We experiment with several one-dimensional synthetic and real-life data set. Due to space con-

straints we only present here the results for the real-life data sets (the performance of the algorithms in the synthetic data sets is qualitatively similar). The `Weather` data set contains meteorological measurements obtained by a station at the university of Washington (data at http://www-k12.atmos.washington.edu/k12/grayskies). This is a one-dimensional data set for which we extracted the following 6 measured quantities: wind speed, wind peak, solar irradiance, relative humidity, air temperature and dewpoint temperature, and present here the results for the wind speed (AirSpeed) and the air temperature (AirTemp), which represent a noisy and a smooth signal, correspondingly. The `Phone` data set includes the total number of long distance calls per minute originating from several states in USA. We here present the results for the states of New York (NY) and Indiana (IN), with NY having large numbers of calls per minute and IN being a state with significantly fewer calls. The presented results for each real-life data set are also indicative of the results for the other measured quantities in these data sets.

Approximation Error Metrics. To compare the accuracy of the studied algorithms we focus on the maximum relative error of the approximation, since it can provide guaranteed error-bounds for the reconstruction of any individual data value. Since the objective function that both studied algorithms try to minimize is the maximum NSE^2 of any data value, for a more direct and clear comparison we present the results for this metric and for both algorithms. The results for the maximum relative error are qualitatively similar to the presented ones.

5.2 Experimental Results

Sensitivity to ϵ. We now evaluate the accuracy and running time of the ϵ-ApproxRV algorithm in comparison to the MinRelVar algorithm, using the real-life data sets. In Figures 3 and 4 we plot the running times for the two algorithms and for the two data sets, correspondingly, as we vary the value of ϵ from 0 to 0.3. We set the domain size for all data sets to 65536, and the synopsis space to 5% of the input size. The ϵ-ApproxRV algorithm is consistently faster than the MinRelVar algorithm in both real-life data sets, often by more than an order of magnitude, and is considerably faster even when the optimal solution is required (ϵ = 0). Unlike the MinRelVar algorithm which may perform multiple lookups of each computed entry,[6] the ϵ-ApproxRV algorithm processes all node entries in a single pass, therefore resulting in significantly faster running times. We also observe in these figures that, with the increase of ϵ, the running time of ϵ-ApproxRV decreases, as

[6] In the MinRelVar algorithm the optimal solution of allocating space B to any node v may be probed for any space allotment $\geq B$ to v's parent node in the error tree.

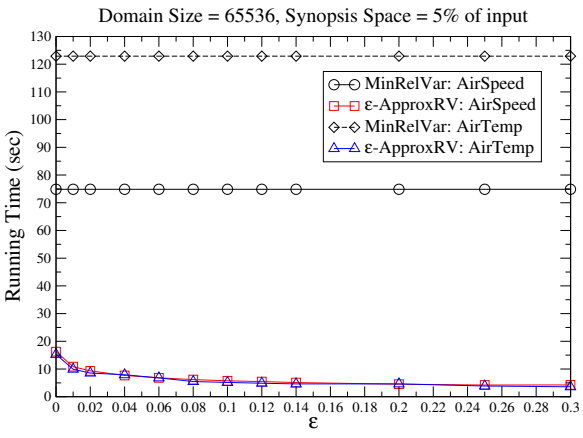

Figure 3: Running Time vs ϵ in Weather data

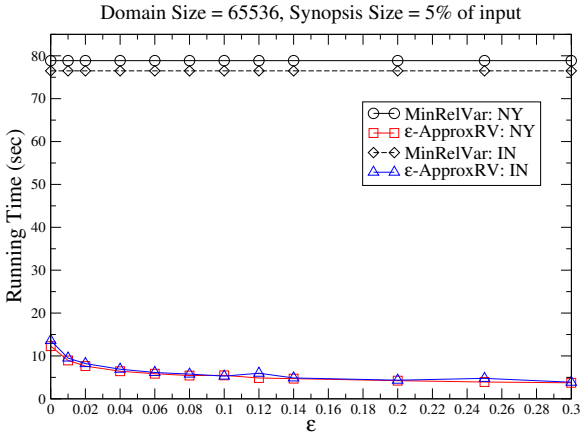

Figure 4: Running Time vs ϵ in Phone data

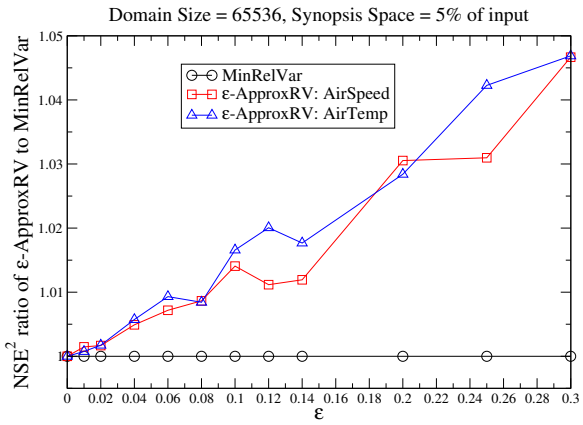

Figure 5: NSE^2 ratio vs ϵ in Weather data

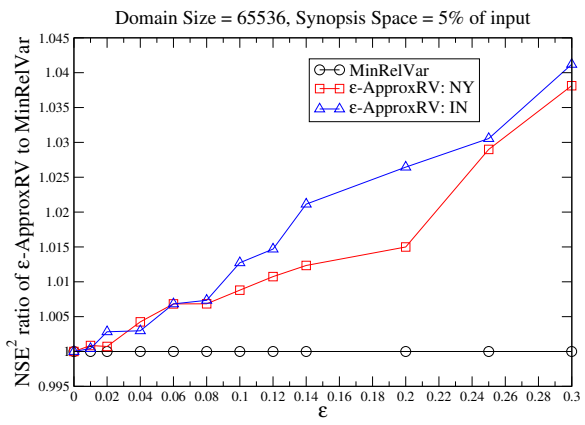

Figure 6: NSE^2 ratio vs ϵ in Phone data

the algorithm effectively prunes a larger number of breakpoints.

The corresponding NSE^2 values for both algorithms are presented in Figures 5 and 6. In order for the reader to be able to observe the difference in the accuracy of the two algorithms, in each figure we plot the ratio of the maximum NSE^2 values produced by the ϵ-ApproxRV algorithm to the corresponding results of the MinRelVar algorithm. The ϵ-ApproxRV algorithm, as expected, always provides solutions that are within the specified error factor ϵ from the optimal solution. It is interesting to note though that in all cases the produced solution is significantly closer to the optimal one (by more than a factor of 5), than the specified ϵ value. This is not surprising, as ϵ represents a worst-case error bound.

Sensitivity to the Domain Size. We now evaluate the accuracy and running time of the ϵ-ApproxRV algo-

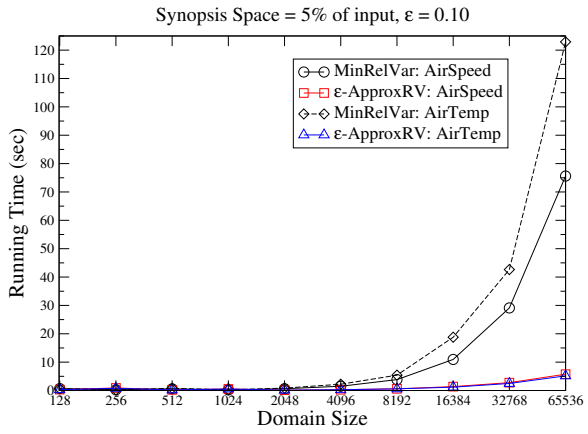

Figure 7: Running Time vs Domain Size in Weather data

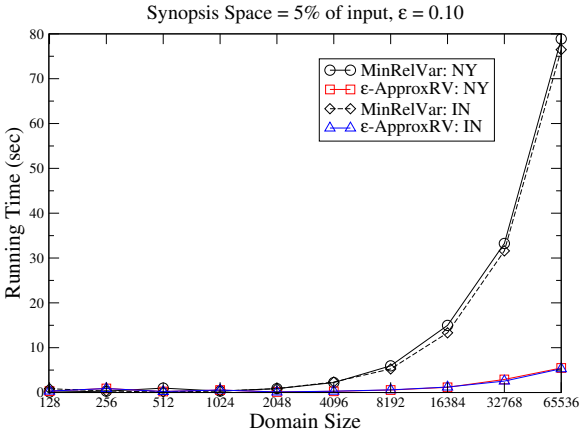

Figure 8: Running Time vs Domain Size in Phone data

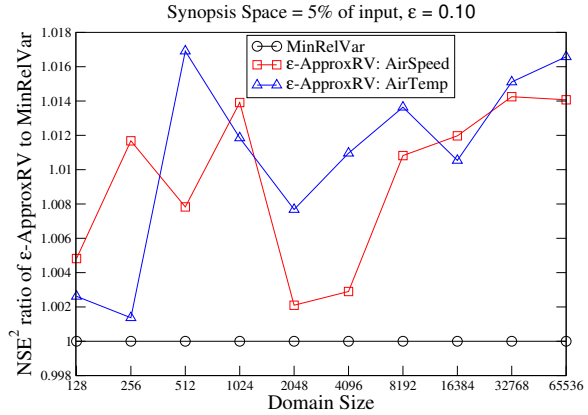

Figure 9: NSE^2 ratio vs Domain Size in Weather data

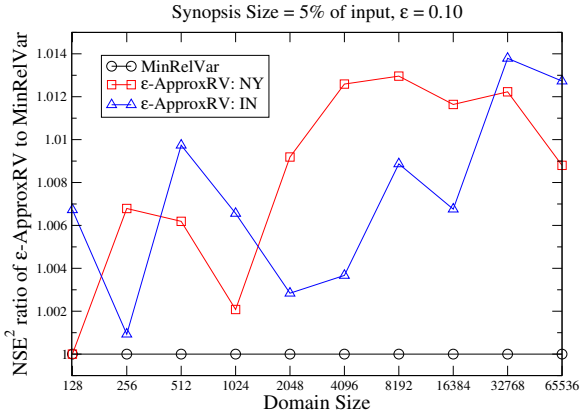

Figure 10: NSE^2 ratio vs Domain Size in Phone data

rithm in comparison to the MinRelVar algorithm, using the real-life data sets, when we vary the domain size of the data sets from 128 to 65536, and plot the resulting running times for the two algorithms in Figures 7 and 8. The synopsis space is set to 5% of the input size, while the value of ϵ is set to 0.10. Again, the ϵ-ApproxRV algorithm significantly outperforms the MinRelVar algorithm, with the savings in running time increasing rapidly as the domain size increases. For large domain sizes, the ϵ-ApproxRV algorithm is up to 23.8 times faster than the MinRelVar algorithm.

In Figures 9 and 10 we plot the corresponding ratios of the maximum NSE^2 values obtained by the two algorithms. Again, the ϵ-ApproxRV algorithm always produced solutions that are significantly closer to the optimal solution (less than 1.7% and 1.4% difference, correspondingly, for the two data sets), than the

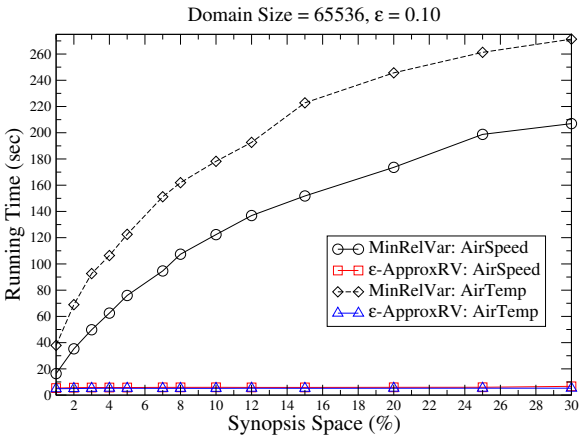

Figure 11: Running Time vs Space in Weather data

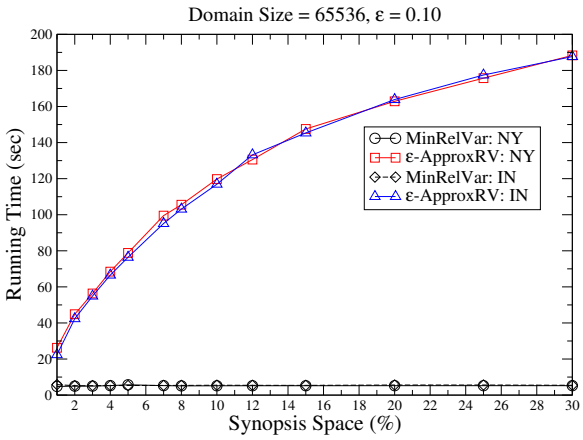

Figure 12: Running Time vs Space in Phone data

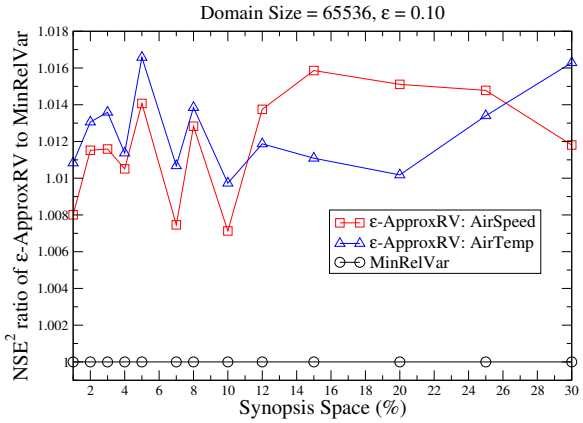

Figure 13: NSE^2 ratio vs Space in Weather data

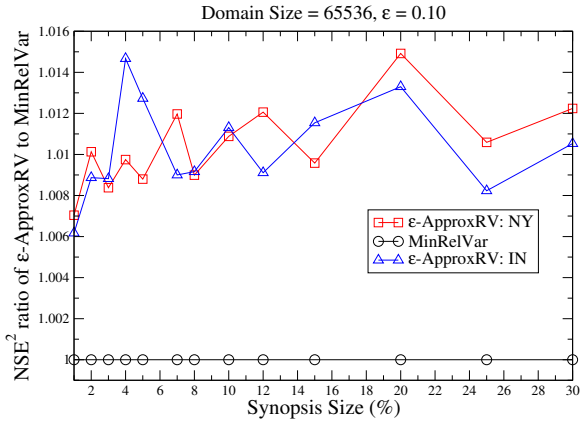

Figure 14: NSE^2 ratio vs Space in Phone data

specified error factor ϵ.

Sensitivity to the Synopsis Space. In Figures 11 and 12 we present the running times for both algorithms and for the real-life data sets, as the synopsis space is varied from 1% to 30% of the size of the input. The domain size is set to 65536, while the value of ϵ is set to 0.10. The running time of the MinRelVar algorithm increases rapidly with the increase in the used synopsis space, while the corresponding running time of the ϵ-ApproxRV algorithm remains practically unaffected. For large synopsis spaces, the ϵ-ApproxRV algorithm is more than two orders of magnitude faster than the MinRelVar algorithm. However, the solutions obtained from the ϵ-ApproxRV algorithm are again very close to the optimal ones (less than 1.7% and 1.5% difference for the two data sets), as we can see in Figures 13 and 14.

6 Conclusions

We have proposed a novel, fast approximation scheme for constructing probabilistic wavelet synopses over large data sets. Our proposed techniques employ a much "sparser" version of previously proposed Dynamic-Programming (DP) solutions, which restricts its search to a carefully chosen, logarithmically-small subset of *"breakpoints"* that cover the entire range of possible space allotments, while always ensuring a maximum relative degradation of $(1 + \epsilon)$ in the quality of the obtained solution. Our experimental evaluation has demonstrated that our approximation algorithm typically provides significantly tighter solutions than the maximum $(1 + \epsilon)$ error factor, while at the same time providing running times that are up to two orders of magnitude smaller than known exact DP solutions.

References

[1] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. "Join Synopses for Approximate Query Answering". In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*.

[2] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. "Approximate Query Processing Using Wavelets". In *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000.

[3] Kaushik Chakrabarti, Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. "Approximate query processing using wavelets". *The VLDB Journal*, 10(2-3):199–223, September 2001. ("Best of VLDB'2000" Special Issue).

[4] Antonios Deligiannakis and Nick Roussopoulos. "Extended Wavelets for Multiple Measures". In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*.

[5] Minos Garofalakis and Phillip B. Gibbons. "Approximate Query Processing: Taming the Terabytes". Tutorial in *27th Intl. Conf. on Very Large Data Bases*, 2001.

[6] Minos Garofalakis and Phillip B. Gibbons. "Wavelet Synopses with Error Guarantees". In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*.

[7] Minos Garofalakis and Phillip B. Gibbons. “Probabilistic Wavelet Synopses”. *ACM Transactions on Database Systems*, 29(1), March 2004. (SIGMOD/PODS Special Issue).

[8] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. “Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries”. In *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001.

[9] Sudipto Guha, Nick Koudas and Kyuseok Shim. “Data-Streams and Histograms”. In *STOC*, 2001.

[10] Peter J. Haas and Arun N. Swami. “Sequential Sampling Procedures for Query Size Estimation”. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*.

[11] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. “Online Aggregation”. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*.

[12] Björn Jawerth and Wim Sweldens. “An Overview of Wavelet Based Multiresolution Analyses”. *SIAM Review*, 36(3):377–412, 1994.

[13] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. “Wavelet-Based Histograms for Selectivity Estimation”. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*.

[14] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. “Dynamic Maintenance of Wavelet-Based Histograms”. In *Proceedings of the 26th International Conference on Very Large Data Bases*, 2000.

[15] Rajeev Motwani and Prabhakar Raghavan. “*Randomized Algorithms*”. Cambridge University Press, 1995.

[16] Rolfe R. Schmidt and Cyrus Shahabi. “ProPolyne: A Fast Wavelet-Based Algorithm for Progressive Evaluation of Polynomial Range-Sum Queries”. In *Proceedings of the Eighth International Conference on Extending Database Technology (EDBT’2002)*.

[17] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. “*Wavelets for Computer Graphics – Theory and Applications*”. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[18] Jeffrey Scott Vitter and Min Wang. “Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets”. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*.