Probabilistic Wavelet Synopses for Multiple Measures

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

The recently proposed idea of *probabilistic wavelet synopses* has enabled their use as a tool for reducing large amounts of data down to compact *wavelet synopses* that can be used to obtain fast, accurate approximate answers to user queries, while at the same time providing guarantees on the accuracy of individual answers. Relatively little attention, however, has been paid to the problem of using wavelet synopses as an approximate query answering tool over complex tabular data sets containing multiple measures, such as those typically found in real-life OLAP applications. In this paper we first demonstrate that for such multi-measure data sets the problem of constructing optimal probabilistic wavelet synopses becomes significantly more complex. We then propose both an exact algorithmic formulation for probabilistic multi-measure wavelet thresholding based on the idea of *Partial-Order Dynamic Programming (PODP)*, and then also introduce a *fast, greedy approximation algorithm* based on the idea of marginal error gains. Our empirical study with both synthetic and real-life data sets validates our approach, demonstrating that (1) our algorithms easily outperform naive approaches based on optimizing individual measures independently, and (2) our greedy thresholding scheme always provides near-optimal and, at the same time, very fast and scalable solutions to the probabilistic wavelet synopsis construction problem.

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) [2, 3, 7, 12].

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 [13, 18]. 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 [14] and approximate query processing over massive relational tables [3, 8, 19] and data streams [10, 15]. 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*. Several research studies [3, 8, 9, 14, 17, 19] have demonstrated that fast and accurate approximate query processing engines can be designed to operate solely over such precomputed compact *wavelet synopses*.

The Haar wavelet decomposition was originally designed with the objective of minimizing the overall root-mean-squared error (i.e., the L_2*-norm average error*) in the data approximation. However, the recent work of [8, 9] on *probabilistic wavelet synopses* also demonstrates their use for optimizing other error metrics, including the *maximum relative error* in the approximate reconstruction of individual data values, which is arguably the most important metric for approximate query answers and, furthermore, enables meaningful, non-trivial *error guarantees* for reconstructed values. 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.

Despite the surge of interest in wavelet-based data reduction and approximation in database systems, relatively little attention has been paid to the application of wavelet techniques to complex tabular data sets *with multiple measures* (multiple numeric entries for each table cell). Such massive, multi-measure tables arise naturally in several application domains, including OLAP environments and time-series analysis/correlation systems. As an example, a corporate sales database may tabulate, for each available product, (1) the number of items sold, (2) revenue and profit numbers for the product, and (3) costs associated with the product, such as shipping and storage costs. Similarly, a network-traffic monitoring system takes readings on each time-tick from a number of distinct elements, such as routers and switches, in the underlying network and

typically several measures of interest need to be monitored (e.g., input/output traffic numbers for each router or switch interface) even for a fixed network element [1]. Both of these types of applications may be characterized not only by the potentially very large domain sizes for some dimensions (e.g. several thousands of time ticks or different products sold), but also by the huge amounts of collected data.

The work in [5] recently introduced the idea of *extended wavelet coefficients* as a flexible, space-efficient storage format for extending conventional wavelet-based summaries to the context of multi-measure data sets. However, the synopsis-construction techniques of [5] can only be used to minimize (for a given space budget) the *weighted sum of the overall L_2-norm errors for each measure*. Still, given the pitfalls and shortcomings of L_2-error-optimized wavelet synopses for building effective approximate query processing engines [8, 9], there is a clear need for more sophisticated wavelet-based summarization techniques for multi-measure data that can be specifically optimized for different error metrics (such as the relative error metric).

Our Contributions. In this paper, we propose the first known algorithms for constructing effective probabilistic wavelet synopses over multi-measure data sets. Our proposed techniques can accommodate a number of different error metrics, including the relative error metric, thus enabling meaningful *error guarantees on the accuracy of the approximation for individual measure values*. Furthermore, by operating on all measures simultaneously, our techniques judicially allocate the available space to all measures based on the difficulty of accurately approximating each one, and exploit storage dependencies among coefficient values to achieve improved storage utilization and, therefore, improved accuracy in data reconstruction than prior techniques that operate on each measure individually. Our contributions are summarized as follows.

• **Probabilistic Wavelets over Multiple Measures: Formulation and Exact PODP Solution.** We formally define the problem of constructing probabilistic wavelet synopses over multi-measure data sets using the space-efficient "extended wavelet coefficient" format of [5]. We demonstrate that utilizing this more involved storage format for coefficients forces non-trivial dependencies between thresholding decisions made across different measures, thus significantly increasing the complexity of probabilistic coefficient thresholding. More specifically, these dependencies cause the key *principle of optimality* based on a *total ordering* of partial solutions [4], that is required by the earlier single-measure DP solutions, to be violated, rendering these techniques inapplicable in the multi-measure setting. Thus, we propose a novel probabilistic thresholding scheme for multi-measure data sets based on the idea of an exact Partial-Order DP (PODP) formulation.

In a nutshell, our PODP solution generalizes earlier single-measure DP schemes [8, 9] to data sets with M measures by using an M*-component vector objective* and an M*-component less-than* partial order to prune sub-problem solutions that cannot possibly be part of an optimal solution.

• **Fast, Greedy Approximate Probabilistic-Thresholding Algorithm.** Given the very high space and time complexities of our PODP algorithm, we also introduce a novel, greedy approximation algorithm (termed GreedyRel) for probabilistic coefficient thresholding over multi-measure data. Briefly, our GreedyRel heuristic exploits the *error-tree structure* [14] for Haar wavelet coefficients in greedily allocating the available synopsis space based on the idea of *marginal error gains*. More specifically, at each step, GreedyRel identifies, for each error subtree, the subset of wavelet coefficients that are expected to give the largest *per-space reduction in the error metric*, and allocates space to the best such subset overall (i.e., in the entire tree). The time and space complexities of our GreedyRel algorithm are only linear in the number of measures involved and the data-set size, and, in fact, are also significantly lower than those of the earlier-proposed DP algorithms for the single-measure case [8, 9]. We must note that the complexities of the algorithms in [8, 9] are, even for the single-measure case, at least quadratic to the domain size, thus yielding our GreedyRel algorithm as the only viable solution, even for the single-measure case, for constructing accurate probabilistic wavelet synopses over large data sets.

• **Experimental Results Verifying the Effectiveness of our Approach.** We present results from an extensive experimental study of our proposed techniques with both synthetic and real-life data sets. Our results clearly validate our approach, demonstrating that (1) our algorithms easily outperform naive approaches based on optimizing individual measures independently, typically producing errors that are up to a factor of 7 smaller than prior techniques; and (2) our greedy thresholding scheme always provides near-optimal and, at the same time, very fast and scalable solutions to the probabilistic wavelet synopsis construction problem.

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. 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 [18].

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 [18]. Suppose 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 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 [18]. Furthermore, the Haar wavelet decomposition can also be

Symbol	Description $i \in \{0, \ldots, N-1\}$, $j \in \{1, \ldots, M\}$, j index/subscript is dropped for $M = 1$
N	Number of data-array cells
D	Data-array dimensionality
M	Number of data-set measures
B	Space budget for synopsis
A, W_A	Input data and wavelet transform arrays
d_{ij}	Data value for i^{th} cell and j^{th} measure of data array
$\hat{d}_{ij}$	Reconstructed data value for i^{th} cell and j^{th} measure
c_{ij}	Haar coefficient at coordinate i for the j^{th} measure
y_{ij}	Retention probability (i.e., fractional storage) for Haar coefficient c_{ij}
C_{ij}	Random variable for Haar coefficient c_{ij}
EC_i	Extended wavelet coefficient at coordinate i
Norm(i, j)	Normalization term for Haar coefficient c_{ij}
`q`	Integer quantization parameter
NSE$(\hat{d}_{ij})$	Normalized standard error for reconstructed $\hat{d}_{ij}$
Var(c_{ij}, y_{ij})	Variance of C_{ij} for a given space y_{ij}
`path`(u)	All non-zero proper ancestors of u in the error tree

Table 1: Notation.

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 [3, 19].

Error Tree and Conventional Wavelet Synopses. A helpful tool for exploring the properties of the Haar wavelet decomposition is the *error tree* structure [14]. 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/coordinate 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. Table 1 summarizes some of the key notational conventions used in this paper; additional notation is introduced when necessary. Detailed symbol definitions are provided at the appropriate locations in the text.

Given a node u in an error tree T, let `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 on the values of coefficients on `path`(d_i); more specifically, we have $d_i = \sum_{c_j \in \texttt{path}(d_i)} \delta_{ij} \cdot c_j$, 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 coordinate 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 [18] by retaining the B largest wavelet coefficients in *absolute normalized value* [18]. 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 [3, 14, 15, 19].

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 recently observed in [8, 9], 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* [16], 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 $\mathrm{E}[C_i] = \lambda_i \cdot y_i +$

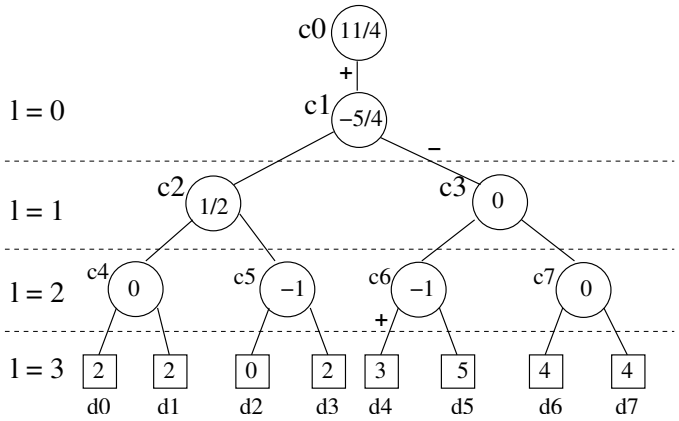

Figure 1: Error tree for example array A.

$0 \cdot (1 - y_i) = c_i$, i.e., the actual coefficient value, while its variance is

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

and the expected size of the synopsis is simply $\text{E}[|\text{synopsis}|] = \sum_{i|c_i \neq 0} y_i = \sum_{i|c_i \neq 0} \frac{c_i}{\lambda_i}$. 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 [8].

The work in [8, 9] proposes 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., $\text{E}[|\text{synopsis}|] \leq B$). 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 [8, 9], the rationale for these probabilistic error metrics is that they are directly related to the *maximum relative error* (with an appropriate *sanity bound* S, whose role is to ensure that relative-error numbers are not unduly dominated by small data values [11, 19]) in the approximation of individual data values based on the synopsis; that is, both the MinRelVar and MinRelBias schemes try to (probabilistically) control the quantity $\max_i\{\frac{|\hat{d}_i - d_i|}{\max\{|d_i|, \text{S}\}}\}$, 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 $\texttt{path}(d_i)$. Bounding the maximum relative error in the approximation also allows for meaningful *error guarantees* to be provided on reconstructed data

values.

To accomplish this, the DP algorithms in [8, 9] seek to minimize the maximum Normalized Standard Error (NSE) in the data reconstruction, defined as

$$\max_i \text{NSE}(\hat{d}_i) = \max_i \frac{\sqrt{\text{Var}(\hat{d}_i)}}{\max\{|d_i|, \mathsf{s}\}},$$

where $\text{Var}(\hat{d}_i) = \sum_{c_j \in \mathtt{path}(d_i)} \text{Var}(j, y_j)$. The algorithms in [9] also naturally extend to multi-dimensional data and wavelets, with a running time of $O(N_z 2^D \mathsf{q} B(\mathsf{q}\log(\mathsf{q}B) + D2^D))$ (N_z being the number of nodes with at least one non-zero coefficient value, N being the maximum domain size and D being the number of dimensions), an overall space requirement of $O(N_z 2^D \mathsf{q} B)$ and an in-memory working-set size of $O(2^D \mathsf{q} B \log N)$. Note that for synopsis spaces $B = O(N_z)$ the above running time and space complexities are at least quadratic to the number of tuples.

Extended Wavelet Coefficients. The wavelet coefficients can be stored as tuples with $D + 1$ fields, where D is the dimensionality of the data array. Each of these tuples contains the D coordinates of the stored wavelet coefficient (one per dimension), which are used to determine the coefficient's support region, and the stored coefficient value. In multi-measure data sets, storage dependencies among different coefficient values may arise. This occurs because two or more coefficient values for different measures may correspond to the same coefficient coordinates, which results in duplicating the storage of these coordinates. This storage duplication increases with the number of the data set's dimensions due to the increased size of the coefficient coordinates.

To alleviate these shortcomings, the work in [5] introduces the notion of an *extended wavelet coefficient*. For a data set comprising M measures, an extended wavelet coefficient is a flexible, space-efficient storage format that can be used to store *any subset* of up to M coefficient values for each combination of coefficient coordinates. Briefly, this is achieved through the use of a *bitmap* of size M, which helps determine exactly the subset of coefficient values that has been stored; thus, the i^{th} bitmap bit is set iff the coefficient for the i^{th} measure has been stored ($1 \leq i \leq M$). More formally, each extended wavelet coefficient is defined as a triplet $\langle C, \beta, V \rangle$ consisting of: (1) The coordinates C of the coefficient; (2) A bitmap β of size M, where the i^{th} bit denotes the existence or absence of a coefficient value for the i^{th} measure; and, (3) The set of stored coefficient values V. We refer to the (coordinates, bitmap) pair for an extended wavelet coefficient as the coefficient's *header*.

3 Probabilistic Wavelets for Multiple Measures

3.1 Problem Formulation and Overview

The work in [5] clearly demonstrated that exploiting storage dependencies among coefficient values can lead to better storage utilization (store more useful coefficient values for the same space bound) and, therefore, improved accuracy to queries. However, the algorithms in [5] can only be applied towards minimizing the overall L_2 error of the approximation, and not for minimizing other error metrics, such as the maximum relative error, which is arguably the most important for providing approximate query answers. On the other hand, while the work in [8, 9] utilized the notion of probabilistic wavelet synopses to propose algorithms that minimize the maximum relative error of the approximation, none of these algorithms can exploit storage dependencies between coefficient values to construct effective *probabilistic wavelet synopses* for multi-measure data sets.

In our work we utilize the notion of the extended wavelet coefficients and the probabilistic wavelet synopses as helpful tools to develop novel algorithms that seek to minimize the maximum relative error of the approximation in multi-measure data sets. To simplify the exposition, we focus our discussion primarily on the one-dimensional case and present the extensions to multi-dimensional wavelets in the Appendix.

Expected Size of Extended Coefficients. The sharing of the common header space (i.e., coordinates + bitmap) among coefficient values introduces non-trivial dependencies in the thresholding process across coefficients for different measures. To be more precise, consider a data set with M measures, and let c_{ij} denote the Haar coefficient value corresponding to the j^{th} measure at coordinate i, and let y_{ij} denote the retention probability for c_{ij} in the synopsis. Also, let EC_i be the extended wavelet coefficient at coordinate i, and let H denote the space required by an extended-coefficient header. (In our discussion, the unit of space is set equal to the space required to store a single coefficient value (e.g., size of a float), and all space requirements are expressed in terms of this unit.) The expected space requirement of the extended coefficient EC_i can be computed as:

$$\mathrm{E}[|EC_i|] = \sum_{j|c_{ij}\neq 0} y_{ij} + H \times (1 - \prod_{j=1}^{M} (1 - y_{ij})). \tag{2}$$

The first summand in the above formula captures the expected space for all (non-zero) individual coefficient values at coordinate i. The second summand captures the expected header overhead. To see this, note that

if at least one coefficient value is stored, then a header space of H must also be allotted. And, of course, the probability of storing ≥ 1 coefficient values is just one minus the probability that none of the coefficients is stored.

Equation (2) clearly demonstrates that the sharing of header space amongst the individual coefficient values c_{ij} for different measures creates a fairly complex dependency of the overall extended-coefficient space requirement on the individual retention probabilities y_{ij}. Given a space budget B for the wavelet synopsis, exploiting header-space sharing and this storage dependency across different measures is crucial for achieving effective storage utilization in the final synopsis. Essentially, this implies that our probabilistic-thresholding strategies for allocating synopsis space cannot operate on each measure individually; instead, space allocation must explicitly account for the storage dependencies across groups of coefficient values (corresponding to different measures). This requirement significantly complicates the design of probabilistic-thresholding algorithms for extended wavelet coefficients.

Problem Statement and Approach. Our goal is to minimize the maximum relative reconstruction error for each individual data value; this would also allow us to provide meaningful *guarantees* on the accuracy of each reconstructed value. More formally, we aim to produce estimates $\hat{d}_{ij}$ of the data values d_{ij}, for each coordinate i and measure index j, such that $|\hat{d}_{ij} - d_{ij}| \leq \epsilon \cdot \max\{|d_{ij}|, \mathsf{S}_j\}$, for given per-measure sanity bounds $\mathsf{S}_j > 0$, where the error bound $\epsilon > 0$ is minimized subject to the given space budget for the synopsis. Since probabilistic thresholding implies that $\hat{d}_{ij}$ is again a random variable, and using an argument based on the Chebyshev bound [8], it is easy to see that *minimizing the overall* NSE *across all measures* (or equivalently, the maximum NSE^2) guarantees a maximum relative error bound that is satisfied *with high-probability*. Thus, we can define our probabilistic-thresholding problem for extended wavelet coefficients as follows.

[Maximum NSE Minimization for Extended Coefficients] Find the retention probabilities y_{ij} for coefficients c_{ij} that *minimize* the maximum NSE for each reconstructed data value across all measures; that is,

$$\text{Minimize} \quad \max_{\substack{i\in\{0,\ldots,N-1\}\\ j\in\{1,\ldots,M\}}} \frac{\sqrt{\text{Var}(\hat{d}_{ij})}}{\max\{|d_{ij}|, \mathsf{s}_j\}} \tag{3}$$

subject to the constraints $0 < y_{ij} \leq 1$ for all non-zero c_{ij} and $\text{E}[|\text{synopsis}|] = \sum_i \text{E}[|EC_i|] \leq B$, where the expected size $\text{E}[|EC_i|]$ of each extended coefficient is given by Equation (2). ∎

We focus on the above maximum NSE minimization problem for multi-measure data in the remainder of this paper. Our algorithms exploit both the error-tree structure of the Haar decomposition and the above-described storage dependencies (Equation (2)) for extended coefficients in order to intelligently assign retention probabilities $\{y_{ij}\}$ to non-zero coefficients within the overall space-budget constraint B. As in [8, 9], our schemes also rely on *quantizing* the space allotments to integer multiples of $1/\mathsf{q}$, where $\mathsf{q} > 1$ is an integer input parameter; that is, we modify the constraint $0 < y_{ij} \leq 1$ to $y_{ij} \in \{\frac{1}{\mathsf{q}}, \frac{2}{\mathsf{q}}, \ldots, 1\}$ in the above problem formulation.

3.2 An Exact Algorithmic Formulation: Partial-Order Dynamic Programming

Consider an input data set with M measures. Our partial-order dynamic programming (PODP) algorithm processes the nodes in the error tree bottom-up and calculates for each node i and each space budget $0 \leq B_i \leq B$ to be allocated to the extended wavelet coefficient values in the node's entire subtree, a collection of incomparable solutions. Each such solution $R[i, B_i]$ is an M*-component vector* of NSE^2 values corresponding to all M measures for the data values in the subtree rooted at node i and assuming a total space of B_i allotted to extended coefficients in that subtree. The goal of the PODP algorithm is, of course, to minimize the maximum component of the vector $R[\mathtt{root}, B]$; that is, minimize $\max_{k=1,\ldots,M}\{R[\mathtt{root}, B]_k\}$.

A key complication in our optimization problem is that, for a given synopsis space budget, these M per-measure NSE values are not independent and cannot optimized individually; this is, again, due to the intricate storage dependencies that arise between the approximation at different measures because of the shared header space (Equation (2)). As already discussed in Section 3.1, it is crucial that our thresholding

algorithms are able to exploit these dependencies to ensure effective synopsis-space utilization. This essentially implies that our thresholding schemes have to treat these M-component NSE vectors *as a unit* during the optimization process.

Let $d_{min_{2i,j}}$ denote the minimum absolute data value in the subtree of node $2i$ and let $\text{Norm}(2i, j) = \max\{d^2_{min_{2i,j}}, \mathsf{S}^2_j\}$ denote a normalization term of the j-th measure for node's i left subtree, with the corresponding normalization term of the right tree defined similarly. We can prove that the j-th component of $R[i, B]$ produced by the optimal assignment of retention probabilities to the coefficient values in the subtree of node i is determined by the minimum absolute data value of measure j in the subtree. This enables us to simplify the minimization problem of Equation 3 by utilizing at each node the normalization terms of its subtrees. The j-th component of $R[i, B]$ at node i for a given retention probability y_{ij} of the c_{ij} coefficient value and solutions $R[2i, b_{2i}]$ and $R[2i + 1, b_{2i+1}]$ from the node's left and right subtrees, can thus be calculated as:

$$\max \begin{cases} \frac{\text{Var}_{(i,y_{ij})}}{\text{Norm}_{(2i,j)}} + R[2i, b_{2i}][j] \\ \frac{\text{Var}_{(i,y_{ij})}}{\text{Norm}_{(2i+1,j)}} + R[2i + 1, b_{2i+1}][j] \end{cases}$$

To ensure optimality, the bottom-up computation of the DP recurrence can not afford to maintain just the locally-optimal partial solution for each subtree. In other words, merely tabulating the $R[i, B]$ vector with the minimum max. component for each internal tree node and each possible space allotment is not sufficient – more information needs to be maintained and explored during the bottom-up computation. As a simple example, consider the scenario depicted in Figure 2 for the case $M = 2$. (Slightly abusing notation, we use $R[2i, B - y]$ and $R'[2i, B - y]$ to denote two possible NSE^2 vectors for space $B - y$ at node $2i$.) To simplify the example, assume that the right child of node i also gives rise to the exact same solution vectors $R[]$ and $R'[]$. In this figure we also depict the normalized variance $\frac{\text{Var}_{(i,y_{ij})}}{\text{Norm}_{(2i,j)}}$ of the coefficient values of node i when total space $y = y_{i1} + y_{i2}$ has been allocated to them and for the data values in the left subtree of node i. It is easy to see that, in this example, even though $R'[2i, B - y]$ is locally-suboptimal at node $2i$ (since its maximal component is larger than the one of $R[]$), it gives a superior overall solution of $[1 + 2, 3 + 0.5] = [3, 3.5]$ at node i when combined with i's local variance vector.

In our PODP algorithm, unlike most DP solutions, the conventional *principle of optimality* based on a *total ordering of partial solutions* [4] is no longer applicable. Thus, locally-suboptimal $R[i, B]$'s (i.e., with large maximum component NSE^2s) cannot be safely pruned since they may, in fact, be part of an optimal

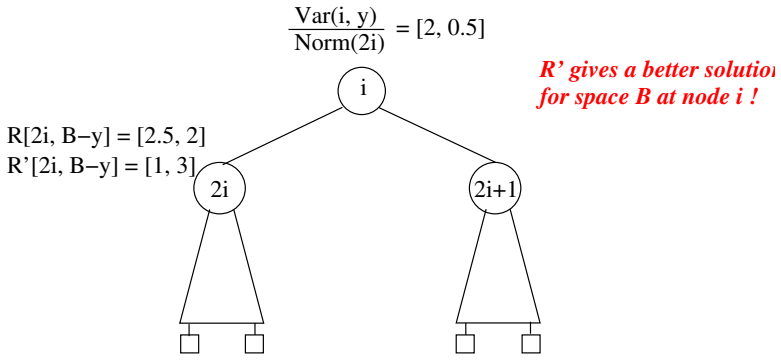

Figure 2: Example for partial-order pruning.

solution higher up in the tree. However, there does exist a safe pruning criterion based on a *partial ordering* of the $R[i, B]$ vectors defined through the M*-component less-than* operator $\preceq_M$, which is defined over M-component vectors u, v as follows:

$$u \preceq_M v \quad \text{if and only if} \quad u_i \leq v_i, \forall i \in \{1, \ldots, M\}.$$

For a given coordinate i and space allotment B, we say that a partial solution $R'[i, B]$ *is covered* by another partial solution $R[i, B]$ if and only if $R[i, B] \preceq_M R'[i, B]$ – it is easy to see that, in this case, $R'[i, B]$ can be safely pruned from the set of partial solutions for the (i, B) combination since, intuitively, $R[i, B]$ can always be used in its place to give an overall solution of at least as good quality.

In our proposed *Partial-Order Dynamic Programming (PODP)*[1] solution to the maximum NSE minimization problem for extended coefficients our partial, bottom-up computed solutions $R[i, B]$ are M-component vectors of per-measure NSE[2] values for coefficient subtrees, and such partial solutions are only pruned based on the $\preceq_M$ partial order. Thus, for each coordinate-space combination (i, B), our PODP algorithm essentially tabulates *a collection* $\mathcal{R}[i, B]$ of incomparable solutions, that represent the "boundary points" of $\preceq_M$,

$$\mathcal{R}[i, B] = \{R[i, b] : \text{ for any other } R'[i, B] \in \mathcal{R}[i, B], \\ R[i, b] \not\preceq_M R'[i, B] \quad \text{and} \quad R'[i, b] \not\preceq_M R[i, B]\}.$$

Of course, for each allotment of space B to the coefficient subtree rooted at node i, our PODP algorithm

[1]Ganguly et al. [6] also discuss PODP in a completely different context, namely designing a System-R-style algorithm for optimizing join orders in parallel database systems.

needs to iterate over all partial solutions computed in $\mathcal{R}[i, B]$ in order to compute the full set of (incomparable) partial solutions for node i's parent in the tree. Similarly, at leaves or intermediate root nodes, we consider all possible space allotments $\{y_{ij}\}$ to each individual measure and estimate the overall space requirements of the extended coefficient using Equation (2). Finally, we note that using an integer parameter $\mathsf{q} > 1$ to quantize possible space allotments introduces some minor complications with respect to the shared header space (e.g., some small space fragmentation) that our algorithm handles.

The main drawback of our PODP-based solution is the dramatic increase in time and space complexity compared to the single-measure case. PODP relies on a much stricter, partial-order criterion for pruning suboptimal solutions which implies that the sets of incomparable partial solutions $\mathcal{R}[i, B]$ that need to be stored and explored during the bottom-up computation can become very large. For instance, in the simple case of a leaf coefficient, it is easy to see that the number of options to consider can be as high as $O(\mathsf{q}^M)$, compared to only $O(\mathsf{q})$ in the single-measure case; furthermore, this number of possibilities can grow extremely fast (in the worst case, *exponentially*) as partial solutions are combined up the error tree.

3.3 A Fast, Greedy Approximation Algorithm

Given the very high running-time and space complexities of our PODP-based solution (described above), we seek to devise an effective approximation algorithm to our maximum NSE minimization problem for extended coefficients. In this section, we propose a very efficient, greedy heuristic algorithm (termed GreedyRel) for this optimization problem. Briefly, GreedyRel tries to exploit some of the properties of dynamic-programming solutions, but allocates the synopsis space to extended coefficients greedily based on the idea of *marginal error gains*. The key idea here is to try, at each step, to allocate additional space to a *subset of extended wavelet coefficients* in the error tree that result in the *largest reduction* in the target error metric (i.e., maximum NSE^2) *per unit of space used*.

Our GreedyRel algorithm relies on three basic operations: (1) Estimating the maximum per-measure NSE^2 values at any node of the error tree; (2) Estimating the best marginal error gain for any subtree by identifying the subset of coefficients in the subtree that are expected to give the largest per-space reduction in the maximum NSE^2; and, (3) Allocating additional synopsis space to the best overall subset of extended coefficients (in the entire error tree). We describe these three operations in detail in the remainder of this section using the notation introduced in the previous sections. Also, let T_{ij} denote the error subtree (for the

j^{th} measure) rooted at c_{ij}.

Estimating Maximum NSE^2 at Error-Tree Nodes. In order to determine the potential reduction in the maximum squared NSE due to extra space, GreedyRel first needs to obtain an estimate for the current maximum NSE^2 at any error-tree node. GreedyRel computes an *estimated maximum* NSE^2 $G[i,j]$ over any data value for the j^{th} measure in the T_{ij} subtree, using the recurrence:

$$G[i,j] = \begin{cases} \max \begin{cases} \frac{\text{Var}(c_{ij},y_{ij})}{\text{Norm}(2i,j)} + G[2i,j] \\ \frac{\text{Var}(c_{ij},y_{ij})}{\text{Norm}(2i+1,j)} + G[2i+1,j] \end{cases} & \text{if } i < N \\ 0 & \text{if } i \geq N. \end{cases}$$

The estimated maximum NSE^2 value is the maximum of two costs calculated for the node's two child subtrees, where each cost sums the estimated maximum NSE^2 of the subtree and the node's variance divided by the subtree normalization term. While one can easily show, as mentioned in Section 3.2, that in the optimal solution the maximum NSE^2 in a subtree will occur for the smallest data value (the proof is based on similar arguments to the single-measure case [9]), the above recurrence is only meant to provide an *easy-to-compute estimate* for a node's maximum NSE^2 (under a given space allotment) that GreedyRel can use.

Estimating the Best Marginal Error Gain for Subtrees. Given an error subtree T_{ij} (for the j^{th} measure), our GreedyRel algorithm computes a subset potSet$[i,j]$ of coefficient values in T_{ij} which, when allotted additional space, are estimated to provide the *largest per-space reduction* of the maximum squared NSE over all data values in the T_{ij} subtree. (Remember that our algorithms allocate the retention probabilities in multiples of $1/\mathsf{q}$, where $\mathsf{q} > 1$.) Let $G[i,j]$ be the current estimated maximum NSE^2 for T_{ij} (as described above), and let $G_{\mathsf{pot}}[i,j]$ denote the *potential* estimated maximum NSE^2 for T_{ij} assuming that the retention probabilities of all coefficient values in potSet$[i,j]$ are increased by a (minimal) additional amount of $1/\mathsf{q}$. Also, let potSpace$[i,j]$ denote the increase in the overall synopsis size, i.e., the cumulative increase in the space for the corresponding *extended* coefficients, when allocating the extra space to the coefficient values in potSet$[i,j]$. We now describe how our GreedyRel algorithm computes potSpace$[i,j]$, and how the best error-gain subsets potSet$[i,j]$ are estimated through the underlying error-tree structure.

Consider a coefficient value $c_{kj} \in$ potSet$[i,j]$. Based on Equation (2), it is easy to see that an increase

of δy_{kj} in the retention probability of c_{kj} results in an increase in the expected-space requirement $\mathrm{E}[|EC_k|]$ of the corresponding extended coefficient EC_k (and, thus, the overall expected synopsis size) of:

$$\delta_j(\mathrm{E}[|EC_k|], \delta y_{kj}) = \delta y_{kj} \cdot (1 + H \times \prod_{p \neq j}(1 - y_{kp})). \tag{4}$$

The total extra space $\mathsf{potSpace}[i,j]$ for all coefficient values in $\mathsf{potSet}[i,j]$ can be obtained by adding the results of Equation (4) for each of these values (with $\delta y_{kj} = \frac{1}{\mathsf{q}}$):

$$\mathsf{potSpace}[i,j] = \sum_{c_{kj} \in \mathsf{potSet}[i,j]} \delta_j(\mathrm{E}[|EC_k|], \frac{1}{\mathsf{q}}).$$

The *marginal error gain* for $\mathsf{potSet}[i,j]$ is then simply estimated as $\mathsf{gain}(\mathsf{potSet}[i,j]) = (G[i,j] - G_{\mathsf{pot}}[i,j])/\mathsf{potSpace}[i,j]$.

To estimate the $\mathsf{potSet}[i,j]$ sets, and the corresponding $G_{\mathsf{pot}}[i,j]$ (and $\mathsf{gain}()$) values at each node, GreedyRel performs a bottom-up computation over the error-tree structure. For a *leaf* coefficient c_{ij}, the only possible choice is $\mathsf{potSet}[i,j] = \{c_{ij}\}$, which can result in a reduction in the maximum NSE^2 if $c_{ij} \neq 0$ and $y_{ij} < 1$ (otherwise, the variance of the coefficient is already 0 and can be safely ignored); in this case, the new maximum NSE^2 at c_{ij} is simply $G_{\mathsf{pot}}[i,j] = \frac{\mathrm{Var}(c_{ij}, y_{ij} + \frac{1}{\mathsf{q}})}{\mathrm{Norm}(i,j)}$.[2] For a *non-leaf* coefficient c_{ij}, GreedyRel considers three distinct cases of forming $\mathsf{potSet}[i,j]$ and selects the one resulting in the largest marginal error gain estimate: (1) $\mathsf{potSet}[i,j] = \{c_{ij}\}$ (i.e., select only c_{ij} for additional storage); (2) $\mathsf{potSet}[i,j] = \mathsf{potSet}[k,j]$, where $k \in \{2i,\, 2i+1\}$ is such that $G[i,j] = G[k,j] + \mathrm{Var}(c_{ij}, y_{ij})/\mathrm{Norm}(k,j)$ (i.e., select the potSet from the child subtree whose estimated maximum NSE^2 determines the current maximum NSE^2 estimate at c_{ij}); and, (3) $\mathsf{potSet}[i,j] = \mathsf{potSet}[2i,j] \cup \mathsf{potSet}[2i+1,j]$ (i.e., select the union of the potSets from both child subtrees). Among the above three choices, GreedyRel selects the one resulting in the largest value for $\mathsf{gain}(\mathsf{potSet}[i,j])$, and records the choice made for coefficient c_{ij} (1, 2, or 3) in a variable ch_{ij}.[3] In order to estimate $\mathsf{gain}(\mathsf{potSet}[i,j])$ for each choice, GreedyRel uses the following estimates for the new maximum NSE^2 $G_{\mathsf{pot}}[i,j]$ at c_{ij} (index k is defined as in case (2) above, and $l = \{2i, 2i+1\} - \{k\}$):

[2] As in [8, 9], in our implementation, we actually cap the contribution of coefficient c_{ij} to the overall variance at c_{ij}^2. This essentially implies (see Section 2) that we only need to consider non-zero allotments $y_{ij} > 1/2$ to coefficient c_{ij}.

[3] It is easy to see that combining the root node c_{ij} with one or both of its child potSets cannot have better marginal error gain than the best of the three options we consider.

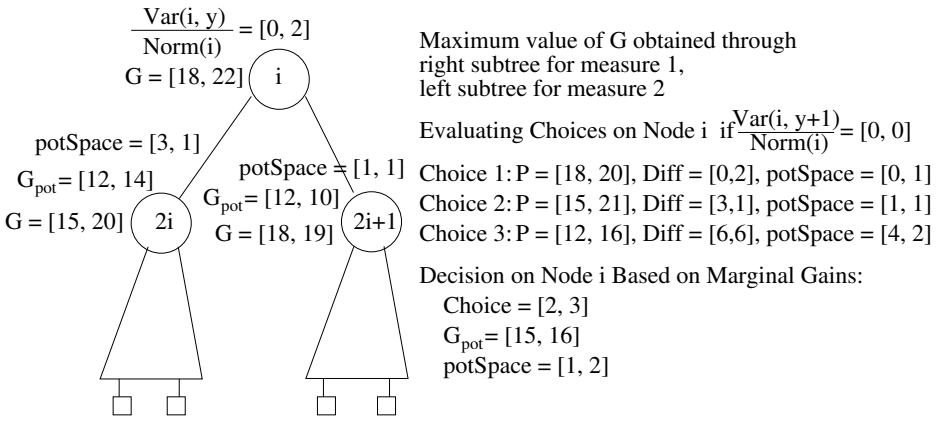

Figure 3: Example for GreedyRel algorithm.

$$G_{\mathsf{pot}}[i,j] = \begin{cases} \max \begin{cases} \frac{\mathrm{Var}(c_{ij}, y_{ij}+\frac{1}{\mathsf{q}})}{\mathrm{Norm}(2i,j)} + G[2i,j] \\ \frac{\mathrm{Var}(c_{ij}, y_{ij}+\frac{1}{\mathsf{q}})}{\mathrm{Norm}(2i+1,j)} + G[2i+1,j] \end{cases} & \mathsf{ch}_{ij} = 1 \\ \max \begin{cases} \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(k,j)} + G_{\mathsf{pot}}[k,j] \\ \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(l,j)} + G[l,j] \end{cases} & \mathsf{ch}_{ij} = 2 \\ \max \begin{cases} \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(2i,j)} + G_{\mathsf{pot}}[2i,j] \\ \frac{\mathrm{Var}(c_{ij}, y_{ij})}{\mathrm{Norm}(2i+1,j)} + G_{\mathsf{pot}}[2i+1,j] \end{cases} & \mathsf{ch}_{ij} = 3 \end{cases}$$

As an example, consider the scenario depicted in Figure 3 for $M = 2$. The figure shows, for each of the children of node i, the computed G, G_{pot}, and potSpace values, along with the value of G and the current normalized variance for node i (assume for simplicity that Norm(2i,j)=Norm(2i+1,j) $\forall j$). The three cases of forming potSet for each measure at node i are enumerated, the corresponding potential reductions (`Diff`) in the estimated maximum NSE^2 value for each measure are calculated, and the choice that results in the largest per-space reduction is selected for each measure. This figure also depicts why it is important to simultaneously increase the retention probabilities of more than one coefficient values. At any node i where the calculated G values through its children are the same, or differ only slightly, for some measure j (as is the case with measure 2 in our example), then any individual assignment of additional space to a coefficient value of that measure below node i would only result in either zero, or very small marginal gains, and would therefore not be selected, independently of how much it would reduce the maximum NSE^2 value through its subtree. This happens because the estimated value of $G[i,j]$ through the other subtree would remain the

same. As the authors of [8] describe, in single-measure data sets the value of G through both subtrees is the same in the optimal solution, thus implying that the above situation is expected to occur very frequently.

An important point to note is that GreedyRel does not need to store the coefficient sets potSet$[i, j]$ at each error-tree node. These sets can be reconstructed on the fly, by traversing the error-tree structure, examining the value of the ch_{ij} variable at each node c_{ij}, and continuing along the appropriate subtrees of the node, until we reach nodes with $\mathsf{ch}_{ij} = 1$.

Distributing the Available Synopsis Space. After completing the above-described steps, our GreedyRel algorithm has computed the estimated current and potential maximum NSE^2 values $G[0, j]$ and $G_{\mathsf{pot}}[0, j]$ (along with the corresponding potSet and potSpace) at the root coefficient (node 0) of the error tree, for each data measure j. Since our overall objective is to minimize the maximum squared NSE among all measures over the entire domain, GreedyRel selects, at each step, the measure $j_{\max}$ with the maximum estimated NSE^2 value at the root node (i.e., $j_{\max} = \arg\max_j\{G[0, j]\}$), and proceeds to allocate additional space of potSpace$[0, j_{\max}]$ to the coefficients in potSet$[0, j_{\max}]$. This is done in a recursive, top-down traversal of the error tree, starting from the root node and proceeding as follows (i denotes the current node index): (1) If $\mathsf{ch}_{ij_{\max}} = 1$, set $y_{ij_{\max}} := y_{ij_{\max}} + \frac{1}{\mathsf{q}}$, (2) If $\mathsf{ch}_{ij_{\max}} = 2$, then recurse to the child subtree T_k, $k \in \{2i, 2i+1\}$ through which the maximum NSE^2 estimate $G[i, j_{\max}]$ is computed at node i, and (3) If $\mathsf{ch}_{ij_{\max}} = 3$, then recurse to both child subtrees T_{2i} and T_{2i+1}; furthermore, after each of the above steps, compute the new G, G_{pot}, potSpace and ch values at node i. These quantities need to be evaluated for all measures because, due to the space dependencies among the coefficient values, the increase of the coefficient value for measure $j_{\max}$ may alter the ch values for the other measures.

Time and Space Complexity. For each of the N error-tree nodes, GreedyRel maintains the variables $G[i, j]$, $G_{\mathsf{pot}}[i, j]$, potSpace$[i, j]$, and ch_{ij}. Thus, the space requirements per node are $O(M)$, resulting in a total space complexity of $O(NM)$.

In the bottom-up initialization phase (Steps 1–6), GreedyRel computes, for each error-tree node, the values of the $G[i, j]$, $G_{\mathsf{pot}}[i, j]$, potSpace$[i, j]$, and ch_{ij} variables (for each measure j). Each of these $O(M)$ calculations can be done in $O(1)$ time, making the total cost of the initialization phase $O(NM)$. Then, note that each time GreedyRel allocates space to a set of K coefficients, the allocated space is $\geq K \times 1/\mathsf{q}$ (see Equation (4)). To reach these K coefficients, GreedyRel traverses exactly K paths of maximum length $O(\log N)$. For each visited node we need to compute the new values of G, G_{pot}, potSpace, and ch, which

Algorithm	Space	Running Time
GreedyRel	$O(N_z M)$	$O(D2^D \times (N_z M + BM\mathsf{q} \log maxD))$
MinRelVar	$O(N_z MB2^D \mathsf{q})$	$O(N_z BM2^D \mathsf{q}(\mathsf{q} \log(\mathsf{q}B) + D2^D))$

Table 2: GreedyRel and MinRelVar complexities.

requires $O(M)$ time. Finding the measure $j_{\max}$ with the maximum estimated NSE^2 value at the root requires time $O(\log M)$ when using a heap structure to store just the $G[0, j]$ values. Thus, GreedyRel distributes space $\geq K \times 1/\mathsf{q}$, in time $O(KM \log N + \log M)$, making the amortized time per-space-quantum $1/\mathsf{q}$ equal to $O(M \log N + \log M/K) = O(M \log N)$. Since the total number of such quanta that we need to distribute is $B\mathsf{q}$, the overall running time complexity of GreedyRel is $O(NM + BM\mathsf{q} \log N)$.

Finally, the GreedyRel algorithms naturally extends to multiple dimensions with a modest increase of $D \times 2^D$ in its running time complexity. These extensions, along with the extensions of PODP to multiple dimensions, can be found in the Appendix. Because the number of non-zero coefficient values in multi-dimensional data sets may be significantly larger than the number of tuples, a thresholding step is needed in this case to limit the space needed by the algorithm. This thresholding step can, of course, also be used in the one-dimensional case to further reduce the running time and space requirements of our GreedyRel algorithm. The work in [9] showed how this step can be performed without introducing any reconstruction bias. Table 2 contains a synopsis of the running time and space complexities of our GreedyRel and the MinRelVar algorithm of [9], where N_z denotes the number of error-tree nodes containing at least one non-zero coefficient value and $maxD$ denotes the maximum domain size among all dimensions.

4 Experimental Study

In this section, we present an extensive experimental study of our proposed algorithms for constructing probabilistic synopses over data sets with multiple measures. The objective of this study is to evaluate both the scalability and the obtained accuracy of our proposed GreedyRel algorithm for a large variety of both real-life and synthetic data sets containing multiple measures. The main findings of our study for the GreedyRel algorithm include:

• **Highly Scalable Solution.** Our GreedyRel algorithm provides a fast and highly-scalable solution for constructing probabilistic synopses over large multi-measure data sets. Unlike earlier schemes (and PODP), GreedyRel scales linearly with the domain size, making it the only viable solution for large real-life data

sets

• **Near Optimal Results.** GreedyRel consistently provides near-optimal solutions when compared to PODP, demonstrating that it constitutes an efficient technique for constructing accurate probabilistic synopses over large multi-measure data sets.

• **Improved Accuracy to Individual Reconstructed Answers.** Compared to earlier approaches that operate on each measure individually, our GreedyRel algorithm significantly reduces the maximum relative error of the approximation, thus being able to offer significantly tighter error guarantees. These improvements are typically by a factor of 2, but in many cases we also observe up to 7 times smaller maximum relative errors.

Techniques and Parameter Settings. Our experimental study compares our GreedyRel and PODP algorithms for constructing probabilistic data synopses over multi-measure data sets, along with a technique, which we will term as IndDP that partitions the available space equally over the measures and then operates on each measure individually by utilizing the dynamic programming MinRelVar algorithm proposed by Garofalakis and Gibbons in [8]. To provide a more fair comparison to the IndDP algorithm, the majority of our experiments includes data sets where all the measured quantities exhibit similar characteristics, thus yielding the uniform partitioning of the synopsis space over all the measures as the appropriate space allocation technique. We note here that we also experimented with the *GreedyL2* algorithm proposed in [5], which is designed to minimize the average sum squared error in multi-measure data sets. However, the *GreedyL2* algorithm consistently exhibited significantly larger errors than our algorithms and is therefore omitted from our experimental results. The only parameter in our algorithms is the quantization parameter q, which is assigned a value of 10 for the GreedyRel and IndDP algorithms, and a smaller value of 4 for the PODP algorithm to reduce its running time. Moreover, the sanity bound of each measure is set to the 5%-quantile value of the measure's data values.

Data Sets. We experiment with several one-dimensional synthetic multi-measure data sets. A Zipfian data generator is used to produce zipfian distributions of various skews (from a low skew of 0.5 to a high skew of 1.5), with the sum of values for each measure set at 200,000. Each zipfian distribution is assigned one of 3 possible shapes: (1) "NoPerm" is the typical zipfian distribution, where smaller domain values are assigned higher values for the measured quantities; (2) "Normal" resembles a bell-shaped normal distribution, with higher (lower) values at the center (endpoints) of the domain; and (3) "PipeOrgan" assigns higher (lower) data values to the endpoints (middle) of the domain. In all cases, the centers of the M distributions are shifted

and placed in random points of the domain. We also consider several different combinations of used zipfian distributions. In the "AllNoPerm" combination, all M of the zipfian distributions have the "NoPerm" shape. Similarly, in the "AllNormal" combination, all M of the zipfian distributions have the "Normal" shape. Finally, in the "Mixed" combination, 1/3 of the M distributions have the "NoPerm" shape, 1/3 have the "Normal" shape, and the remaining had the "PipeOrgan" shape. The results presented in this section are indicative of the multiple possible combinations of our parameters.

In our experimental study we also use real-life data sets. The `Weather` data set contains meteorological measurements obtained by a station at the university of Washington (data found at `http://www-k12.-atmos.washington.edu/k12/grayskies`). This is a one-dimensional data set for which we extract the following 6 measured quantities: wind speed, wind peak, solar irradiance, relative humidity, air temperature and dewpoint temperature. The `Phone` data set includes the total number of long distance calls per minute originating from 6 states (CA, GA, NJ, NY, TX, WA).

Approximation Error Metric. In all cases, 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, and is the error metric that our algorithms try to minimize.

Comparing PODP and GreedyRel**.** We now evaluate the accuracy and running time of the GreedyRel algorithm in comparison to the PODP algorithm. In Figures 4, 5 and 6 we plot the running time and the maximum and average relative errors, correspondingly, for the two algorithms and for the Weather data set when we vary the synopsis space from 10 to 50 units of space (recall that the unit of space is the size of each data value, i.e., sizeof(float)). In this experiment we only use from the Weather data the three most difficult to approximate measures. The domain size of the data set is set to 128. Note that in all our plots depicting the running time of algorithms, the Y axis is logarithmic. Clearly, the running time of the PODP algorithm does not scale well with the size of the data synopsis, even for such a small data set. For example, for a synopsis size of 50 space units, the PODP algorithm requires more than 2 hours to complete, while the GreedyRel algorithm required just a few milliseconds. However, as Figures 5 and 6 demonstrate, the GreedyRel algorithm provides near-optimal solutions in all cases.

In Figure 7 we present the corresponding running times for both algorithms, as the domain size is increased from 64 to 512. From the weather data set we again extract just three measures, and set the synopsis space to always be 5% of the size of the input. Again, the running time performance of PODP is disappointing. For a domain size of 512, its running time exceeds 14 hours. Finally, as Figure 8

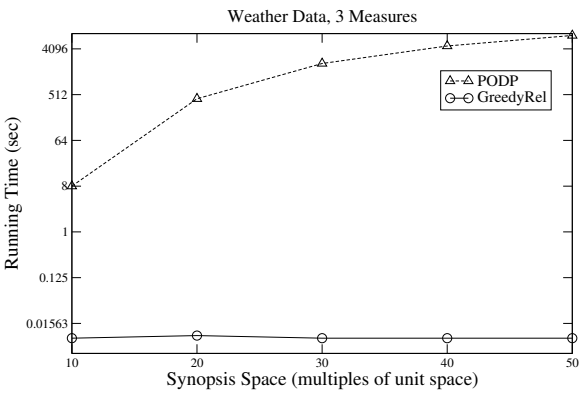

Figure 4: Running Time vs Space

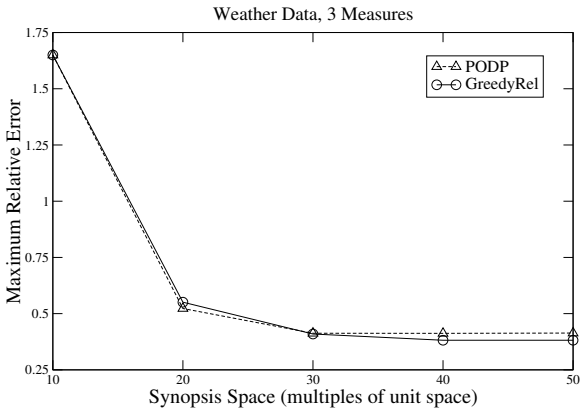

Figure 5: Maximum Relative Error

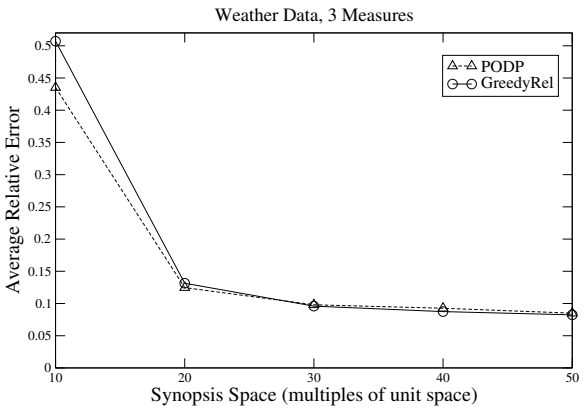

Figure 6: Average Relative Error

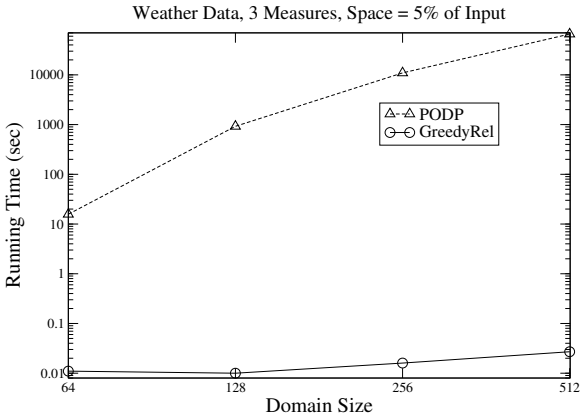

Figure 7: Running Time vs Domain

demonstrates, the running time of PODP increases exponentially with the number of the data set measures. Note that for data sets with 4 or more measures, the PODP does not terminate within one day. It is easy to see that the PODP algorithm cannot be used but for toy-like data sets. On the other hand, the GreedyRel algorithm provides near-optimal solutions in all tested cases, while exhibiting small running times.

Running Time Comparison of GreedyRel **and** IndDP**.** We now compare GreedyRel and IndDP in terms of their running time. In Figure 9 we plot the running times of the IndDP and GreedyRel algorithms for the Weather data set (all 6 measures were included) as the domain size is increased from 128 to 524288. The synopsis size is always set to 5% of the input data. The IndDP algorithm is considerably slower than the GreedyRel algorithm (3 orders of magnitude slower for domain size 131,072), with their difference increasing rapidly with the increase of the domain size. Note that while the GreedyRel algorithm scales linearly with the increase in the domain size (doubling the domain size doubles the running time), the IndDP

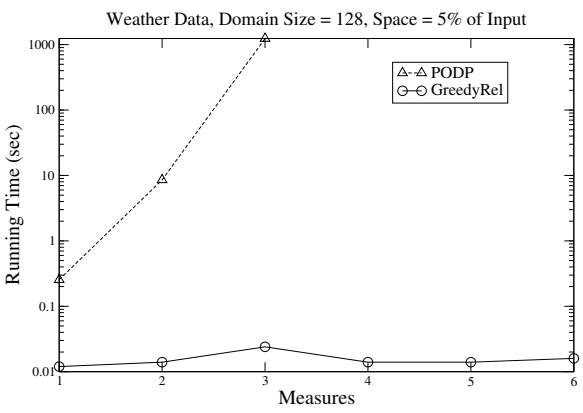

Figure 8: Running Time vs Measures

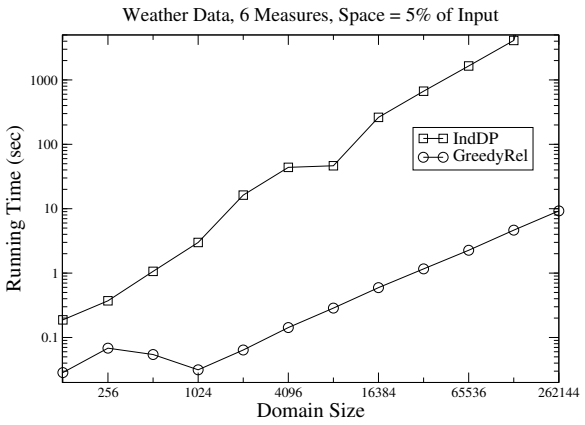

Figure 9: Running Time vs Domain Size

algorithm grows much faster every time the domain size is doubled. This is of course consistent with the running time complexity of the IndDP algorithm (see Section 2), since when the domain size is doubled, the synopsis space is doubled as well. Moreover, the large memory requirements ($O(NB\mathsf{q})$) of the IndDP algorithm prevented it from terminating for domain sizes larger than 131,072 (the main memory of our machine was 512MB). Thus, the linear scalability of the GreedyRel algorithm to the domain size, in terms of both its running time and its memory requirements, constitutes it as the only viable technique (except for small data sets) for providing tight error guarantees, not only on multi-measure data sets, but also on single-measure data sets, since both the GreedyRel and IndDP algorithms scale in a similar way for such data sets. Moreover, as we will demonstrate in this section, the GreedyRel algorithm, which utilizes the extended wavelet coefficients to store the selected coefficient values, also outperforms the IndDP algorithm in terms of the obtained accuracy of the data synopsis. The improved accuracy is attributed to the improved storage utilization achieved by the use of extended wavelet coefficients, and the ability of our GreedyRel algorithm to exploit the underlying storage dependencies.

Accuracy Comparison of GreedyRel **and** IndDP **in Synthetic Data Sets.** For our synthetic data sets, we use a domain size of 256, and present the obtained accuracy in terms of the maximum error of the approximation for the GreedyRel and IndDP algorithms and six representative combinations of synthetic data sets. These six combinations arise from considering zipfian distributions with skew 0.6 and 1, along with all the other possible combinations of the used zipfian distributions ("AllNoPerm", "AllNormal" and "Mixed"). The synthetic data sets in this section contain 6 measures/distributions.

We first consider the six possible combinations arising from distributions having skew equal to 1. In Figures 10, 11 and 12 we plot the maximum relative errors for the GreedyRel and IndDP algorithms, as

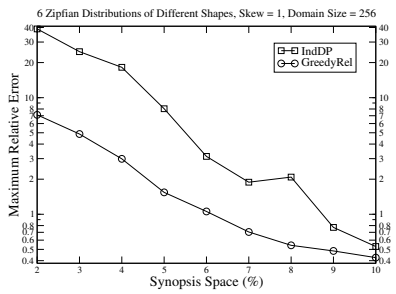

Figure 10: Skew 1, “Mixed”

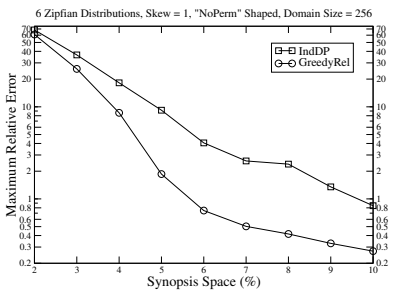

Figure 11: Skew 1, “AllNoPerm”

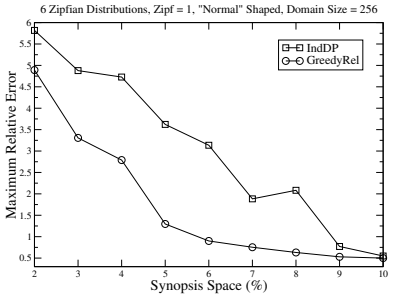

Figure 12: Skew 1, “AllNormal”

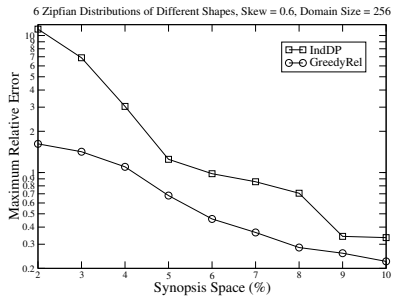

Figure 13: Skew 0.6, “Mixed”

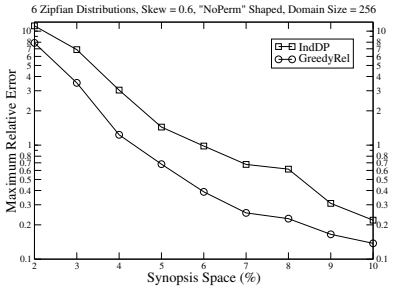

Figure 14: Skew 0.6, “AllNoPerm”

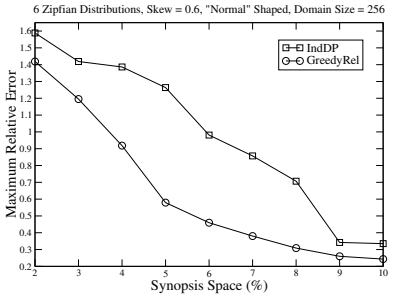

Figure 15: Skew 0.6, “AllNormal”

the synopsis space is varied from 2% to 10% of the input data size, and for the “Mixed”,“AllNoPerm” and “AllNormal” (in the specific order) selection of zipfian distribution shapes. Note that the Y axis for the “AllNoPerm” and “Mixed” cases is logarithmic, due to the large maximum errors observed in this case, mainly by the IndDP algorithm. Intuitively, this occurs because the shifting of some distribution centers in this case results in the largest values of the data set being adjacent to the smallest values, thus requiring several coefficient values to capture this large difference of the values. As we can see, the GreedyRel algorithm produces more accurate results than the IndDP algorithm, with the differences being more significant in the “AllNoPerm” and “Mixed” cases (recall that the Y axis is logarithmic in these 2 cases). Even though none of the techniques produces tight error bounds for such a large data skew value and for small data synopses, the improvements achieved by the GreedyRel algorithm are very significant in each combination of used zipfian distributions. For each combination, GreedyRel produces, correspondingly, up to 6.1, 5.7 and 3.5 times smaller maximum relative errors than IndDP.

Similar results are also observed for the six combinations of synthetic data sets, arising from setting the skew of the distributions to 0.6. In Figures 13, 14 and 15 we show the corresponding results for the “Mixed”, “AllNoPerm” and “AllNormal” combinations of used data distributions (note the logarithmic Y

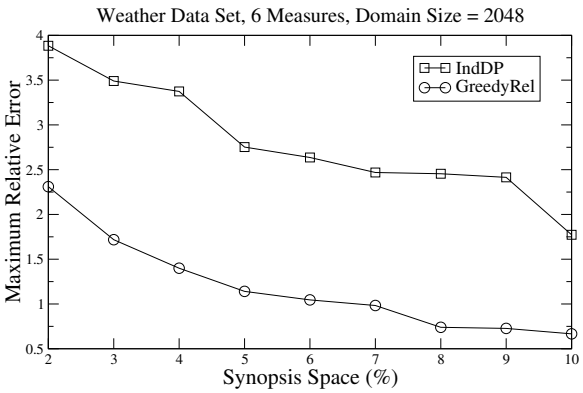

Figure 16: Weather Data Set

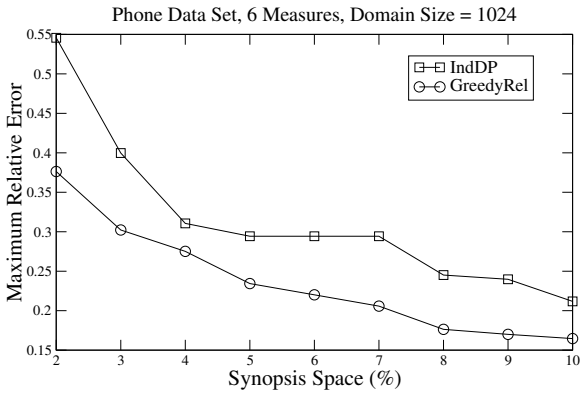

Figure 17: Phone Data Set

axis in the "AllNoPerm" and "Mixed" cases). The maximum relative errors in this case are significantly smaller for all methods. However, the GreedyRel algorithm is still able to provide substantial more tight error bounds, up to 6.9, 2.7 and 2.3 times smaller than IndDP.

Accuracy Comparison of GreedyRel and IndDP in Real Data Sets. In Figures 16 and 17 we plot the maximum relative errors, respectively, for the Weather and Phone data sets, as we vary the size of the synopsis, and for domain sizes of 2048 and 1024, respectively. As we can see, the benefits of the GreedyRel algorithm continue to be significant in all cases. In the Weather data set, the GreedyRel algorithm provided up to 3.5 times tighter error bounds than the IndDP algorithm (and commonly at least a 2-fold improvement), while in the Phone data the corresponding error bounds were up to 1.75 times tighter.

5 Conclusions

We have proposed novel, effective techniques for building probabilistic wavelet synopses over multi-measure data sets. Our techniques seek to minimize, given a storage constraint, the maximum relative error of reconstructing any data value among all measures. We have demonstrated the difficulty of the problem compared to the single-measure case, and have proposed a partial-order dynamic programming (PODP) solution. Given the extremely high time and space complexities of PODP, we have also introduced a very fast and scalable approximate algorithm, which greedily allocates synopsis space based on the idea of marginal error gains. Our experimental evaluation has demonstrated that our GreedyRel algorithm exhibits near-optimal solutions, while at the same time outperforming prior techniques based on optimizing each measure independently. Perhaps more importantly, GreedyRel constitutes the only viable solution, even in the single-measure

case, for constructing accurate probabilistic wavelet synopses over large data sets.

References

[1] "NetFlow Services and Applications". Cisco Systems White Paper (`http://www.cisco.com/`), 1999.

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. "Join Synopses for Approximate Query Answering". In *ACM SIGMOD 1999*.

[3] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. "Approximate Query Processing Using Wavelets". In *VLDB 2000*.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *"Introduction to Algorithms"*. MIT Press, 1990.

[5] A. Deligiannakis and N. Roussopoulos. "Extended Wavelets for Multiple Measures". In *ACM SIGMOD 2003*.

[6] S. Ganguly, W. Hasan, and R. Krishnamurthy. "Query Optimization for Parallel Execution". In *ACM SIGMOD 1992*.

[7] M. Garofalakis and P. B. Gibbons. "Approximate Query Processing: Taming the Terabytes". Tutorial in *VLDB 2001*.

[8] M. Garofalakis and P. B. Gibbons. "Wavelet Synopses with Error Guarantees". In *ACM SIGMOD 2002*.

[9] M. Garofalakis and P. B. Gibbons. "Probabilistic Wavelet Synopses". *ACM Transactions on Database Systems*, 29(1), March 2004.

[10] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries". In *VLDB 2001*.

[11] P. J. Haas and A. N. Swami. "Sequential Sampling Procedures for Query Size Estimation". In *ACM SIGMOD 1992*.

[12] J. M. Hellerstein, P. J. Haas, and H. J. Wang. "Online Aggregation". In *ACM SIGMOD 1997*.

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

[14] Y. Matias, J. S. Vitter, and M. Wang. “Wavelet-Based Histograms for Selectivity Estimation”. In *ACM SIGMOD 1998*.

[15] Y. Matias, J. Scott Vitter, and M. Wang. “Dynamic Maintenance of Wavelet-Based Histograms”. In *VLDB 2000*.

[16] R. Motwani and P. Raghavan. *“Randomized Algorithms”*. Cambridge University Press, 1995.

[17] R. R. Schmidt and C. Shahabi. “ProPolyne: A Fast Wavelet-Based Algorithm for Progressive Evaluation of Polynomial Range-Sum Queries”. In *EDBT 2002*.

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

[19] J. S. Vitter and M. Wang. “Approximate Computation of Multi- dimensional Aggregates of Sparse Data Using Wavelets”. In *ACM SIGMOD 1999*.

Appendix

Pseudocode for GreedyRel Algorithm

A pseudocode description of our GreedyRel algorithm is depicted in Figure 18. The algorithm is straightforward, based on the discussion of Section 3.3. Note that in the later steps of the algorithm, the available synopsis space may become smaller than potSpace$[i, j_{\max}]$; in this case, rather than recursing on both child subtrees of a node (when $\mathsf{ch}_{ij_{\max}} = 2$), GreedyRel first recurses on the child causing the maximum estimated squared NSE, and then recurses on the other child with any remaining space (Steps 12–16 of traverse).

Extensions to Multi-Dimensional Wavelets

We now discuss the key ideas for extending our techniques to multi-dimensional data. For a detailed discussion of multi-dimensional wavelets we refer the reader to the analysis presented in [9]. 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.

Extending PODP. Our PODP algorithm for multi-dimensional data sets generalizes the corresponding multi-dimensional MinRelVar strategy in [9], in a way analogous to the one-dimensional case. In a nutshell, PODP needs to consider, at each internal node of the error tree, the optimal allocation of space to the $\leq 2^D - 1$ wavelet coefficients of the node and its $\leq 2^D$ child subtrees. The extension of PODP to multi-dimensional data sets is therefore a fairly simple adaptation of the multi-dimensional MinRelVar algorithm. However, as discussed in Section 3.2, PODP needs to maintain, for each node i and each possible space allotment B, a *collection* $\mathcal{R}[i, B]$ of incomparable solutions. This requirement, once again, makes the time/space requirements of PODP significantly higher than those of MinRelVar.

Extending GreedyRel. The first modification involved in extending our GreedyRel algorithm to multi-dimensional data sets has to do with the computation of $G[i, j]$, which now involves examining the estimated NSE^2 values over $\leq 2^D$ child subtrees and maintaining the maximum such estimate. Let $\mathcal{S}(i)$ denote the set of the $\leq 2^D - 1$ coefficients of node i, and let $i_1, \ldots, i_p$ be the indexes of i's child nodes in the error tree.

Then,

$$G[i,j] = \begin{cases} \max \begin{cases} \sum_{c_k \in \mathcal{S}(i)} \frac{\text{Var}(c_{kj}, y_{kj})}{\text{Norm}(i_1, j)} + G[i_1, j] & \\ \ldots & i < N \\ \sum_{c_k \in \mathcal{S}(i)} \frac{\text{Var}(c_{kj}, y_{ij})}{\text{Norm}(i_p, j)} + G[i_p, j] & \end{cases} \\ \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i \geq N \end{cases}$$

The only other necessary modification involves the estimation of marginal error gains at each node. In Section 3.3, we consider a total of three possible choices for forming potSet$[i, j]$ for each (node, measure) combination. Now, each node has up to 2^D child subtrees, resulting in a total of $2^D + 1$ possible choices of forming potSet$[i, j]$. The first choice is to increase the retention probability for measure j of one of the $\leq 2^D - 1$ coefficients in node i. In this case, we simply include in potSet$[i, j]$ the coefficient in node i that is expected to exhibit the largest marginal gain for measure j. For each of the remaining 2^D possible choices of forming potSet$[i, j]$, the k-th choice ($1 \leq k \leq 2^D$) considers the marginal gain of increasing the retention probabilities in the child subtrees through which the k maximum NSE^2 values occur, as estimated in the right-hand side of the above equation for $G[i, j]$. At each node, the computation of $G_{\text{pot}}[i, j]$, potSpace$[i, j]$, and ch_{ij} incurs a worst-case time cost of $O(D \times 2^D)$ due to the possible ways of forming potSet$[i, j]$, and the required sorting operation of 2^D quantities. 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 running time complexity of GreedyRel becomes $O(D \times 2^D \times (NM + BM\mathsf{q} \log maxD))$. Note, of course, that in most real-life scenarios using wavelet-based data reduction, the number of dimensions is typically a small constant (e.g., 4–6) and, most importantly, that the number of tuples can be exponential ($O(N^D)$) to the maximum domain size N.

Improving the complexity of GreedyRel**.** 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 [9] 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 [9] 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 GreedyRel 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 afore-mentioned 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 G, G_{pot} values 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 G, G_{pot} values in the descendant nodes of k, since they will be zero. 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 G, G_{pot} values 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 G, G_{pot} values of node n needs to be calculated, using as input the G, G_{pot} values of its non-zero descendant tree nodes. It is easy to demonstrate that at most $N_z - 1$ such nodes may exist. Thus, the GreedyRel algorithm will need to calculate the G, G_{pot} values in at most $O(2N_z - 1)$=$O(N_z)$ nodes, thus yielding running time and space complexities of $O(D \times 2^D \times (N_z M + BM\mathsf{q} \log maxD))$ and $O(N_z M)$, 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. This requires an additional $O(N_z \log N_z)$ time for the sorting 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.

procedure GreedyRel(W_A, B, q, S)
Input: $N \times M$ array W_A of Haar wavelet coefficients; space constraint B; quantization parameter $q > 1$; vector of per-measure sanity bounds S.
Output: Array y of retention probabilities y_{ij} for all $N \times M$ coefficients.
begin
1. **for** $i := N - 1$ **downto** 0 **do** // *traverse error tree bottom-up*
2. **for** $j := 1$ **to** M **do**
3. $y_{ij} = 0$
4. Compute $G[i,j]$, $G_{\text{pot}}[i,j]$, potSpace$[i,j]$, and ch$_{ij}$
5. **endfor**
6. **endfor**
7. spaceLeft = B
8. **while** (spaceLeft > 0) **do**
9. $j_{\text{max}} := \arg\max_j \{G[0,j]\}$
10. occupiedSpace := traverse(0, j_{max}, q, y, spaceLeft)
11. spaceLeft := spaceLeft - occupiedSpace
12. **if** (occupiedSpace = 0) **then return**(y) //*not enough space*
13. **endwhile**
14. **return**(y)
end

procedure traverse(i, j, q, y, spaceLeft)
Input: Index i of error-tree node; measure j chosen for space allocation; quantization parameter q; array y of current retention probabilities; maximum synopsis space to allocate (spaceLeft).
Output: Space allocated to the T_{ij} subtree at this step.
begin
1. allocatedSpace := 0
2. **if** (ch$_{ij}$ = 1) **then**
3. neededSpace := $\delta_j(\text{E}[|EC_i|], 1/q)$ // *see Equation (4)*
4. **if** (neededSpace $\leq$ spaceLeft) **then**
5. $y_{ij} := y_{ij} + 1/q$
6. allocatedSpace := neededSpace
7. **endif**
8. **else if** (ch$_{ij}$ = 2) **then**
9. Find index k of child subtree through which $G[i,j]$ occurs
10. allocatedSpace := traverse(k, j, q, y, spaceLeft)
11. **else**
12. Find index k of child subtree through which $G[i,j]$ occurs
13. Let l be the index of the other subtree
14. allocatedSpace := traverse(k, j, q, y, spaceLeft)
15. **if** (spaceLeft $>$ allocatedSpace) **then**
16. allocatedSpace += traverse(l, j, q, y, spaceLeft-allocatedSpace)
17. **endif**
18. Recompute the node's G, G_{pot}, potSpace, and ch values
19. **return**(allocatedSpace)
end

Figure 18: GreedyRel Algorithm Pseudocode.