Extended Wavelets for Multiple Measures

ANTONIOS DELIGIANNAKIS
University of Athens
MINOS GAROFALAKIS
Intel Research
and
NICK ROUSSOPOULOS
University of Maryland

Several studies have demonstrated the effectiveness of the Haar wavelet decomposition 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. Although originally designed for minimizing the overall mean-squared (i.e., L_2-norm) error in the data approximation, recently proposed methods also enable the use of Haar wavelets in minimizing other error metrics, such as the relative error in data value reconstruction, which is arguably the most important for approximate query answers. Relatively little attention, however, has been paid to the problem of using wavelet synopses as an approximate query answering tool over complex tabular datasets containing *multiple measures*, such as those typically found in real-life OLAP applications. Existing decomposition approaches will either operate on each measure individually, or treat all measures as a vector of values and process them simultaneously. As we demonstrate in this article, these existing *individual* or *combined* storage approaches for the wavelet coefficients of different measures can easily lead to suboptimal storage utilization, resulting in drastically reduced accuracy for approximate query answers. To address this problem, in this work, we introduce the notion of an *extended* wavelet coefficient as a flexible, efficient storage method for wavelet coefficients over multimeasure data. We also propose novel algorithms for constructing effective (optimal or near-optimal) extended wavelet-coefficient synopses under a given storage constraint, for both sum-squared error and relative-error norms. Experimental results with both real-life and synthetic datasets validate our approach, demonstrating that our techniques consistently obtain significant gains in approximation accuracy compared to existing solutions.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Wavelets, data synopses, approximate query processing

Authors' addresses: A. Deligiannakis, University of Athens, 22A Klavdiou Galinou Street, 15127 Melissia, Athens, Greece; email: adeli@di.uoa.gr; M. Garofalakis, Intel Research Berkeley, 2150 Shattuck Avenue, Penthouse Suite, Berkeley, CA 94704; email: minos.garofalakis@intel.com; N. Roussopoulos, Department of Computer Science, University of Maryland, College Park, MD 20742; email: nick@cs.umd.edu.

ACM Reference Format:
Deligiannakis, A., Garofalakis, M., and Roussopoulos, N. 2007. Extended wavelets for multiple measures. ACM Trans. Datab. Syst. 32, 2, Article 10 (June 2007), 52 pages. DOI = 10.1145/1242524.1242527 http://doi.acm.org/10.1145/1242524.1242527

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 "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) [Acharya et al. 1999; Chakrabarti et al. 2001; Garofalakis and Gibbons 2001; Hellerstein et al. 1997]. In recent years, several different methods have been explored for reducing massive data collections into the compact data synopses employed in the approximate-query processing engine; these include *random sampling*, *histograms*, and, more recently, *wavelets*.

Haar wavelets are a mathematical tool for the hierarchical decomposition of functions, with several successful applications in signal and image processing [Jawerth and Sweldens 1994; Stollnitz et al. 1996]. A number of recent studies have also demonstrated the effectiveness of the Haar wavelet decomposition as a data reduction tool for database problems, including selectivity estimation [Matias et al. 1998] and approximate query processing over massive relational tables [Chakrabarti et al. 2001; Garofalakis and Gibbons 2004; Vitter and Wang 1999] and data streams [Gilbert et al. 2001; Matias et al. 2000]. Briefly, the key idea is to apply the decomposition process over an input dataset, along with a thresholding procedure, in order to obtain a compact data synopsis comprised of a selected small set of *Haar wavelet coefficients*. The results of several research studies [Chakrabarti et al. 2001; Garofalakis and Gibbons 2004; Matias et al. 1998; Schmidt and Shahabi 2002; Vitter and Wang 1999] have demonstrated that fast and accurate approximate query processing engines can be designed to operate solely over such compact *wavelet synopses*. Furthermore, even though Haar wavelet decomposition was originally designed with the objective of minimizing the overall mean-squared error (i.e., the *L_2-norm error*) in the data approximation, recent work [Garofalakis and Gibbons 2004; Garofalakis and Kumar 2005] has also demonstrated their use in optimizing *relative-error* metrics. Relative errors are arguably the most important metrics for approximate-query answers and can also enable meaningful, nontrivial *error guarantees* for reconstructed values (by bounding the *maximum relative-error* in the approximate reconstruction of individual data values [Garofalakis and Gibbons 2004; Garofalakis and Kumar 2005]).

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 datasets with *multiple measures* (multiple numeric entries for each table cell). Such massive, multimeasure 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, real-life applications that monitor continuous time-series typically have to deal with several readings (measures) that evolve over time; for example, a network-traffic monitoring system takes readings on each time-tick from a number of distinct elements (i.e., 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 [Cisco 1999].

Traditionally, two obvious strategies, termed *Individual* and *Combined*, have been employed when adapting wavelet-based methods over such multimeasure datasets. The Individual algorithm performs wavelet decomposition on each of the individual measures, and stores the important coefficients for each measure separately. On the other hand, the Combined algorithm performs a joint wavelet decomposition on the multimeasure dataset by treating all the measures as a *vector* of values and, at the end, determines a subset of vectors of coefficient values to retain in the synopsis. As we demonstrate, such obvious *individual* or *combined* approaches can lead to poor synopsis-storage utilization and suboptimal solutions, even in very simple cases. Due to the nature of wavelet decomposition and the possible correlations across different measures, there are many scenarios in which multiple—but not necessarily all—wavelet coefficients at the same coordinates have large values, and are thus beneficial to retain, for instance, in an L_2-optimized synopsis. In such cases, the Individual algorithm essentially replicates the storage of the shared coordinates multiple times, wasting valuable synopsis storage. The Combined algorithm, on the other hand, stores *all* coefficient values sharing the same coordinates, thus wasting space by storing small, unimportant values for certain measures.

Our Contributions. In this work, we propose a novel approach for effectively adapting wavelet-based data reduction methods to multimeasure datasets through the use of *extended wavelet coefficients*. Briefly, an extended wavelet coefficient can store multiple coefficient values for different—but not necessarily all—measures. The end result is a flexible, space-efficient storage scheme that can eliminate the disadvantages of both the Individual and Combined algorithms discussed earlier. We then consider the problem of constructing effective extended wavelet-coefficient synopses (under a given storage constraint) optimized for the: (1) *weighted sum-squared error*, and (2) *relative error* in the approximate data reconstruction. Our synopsis construction problems are natural generalizations of the corresponding problems for conventional (i.e., L_2-error) wavelet synopses [Vitter and Wang 1999; Chakrabarti et al. 2001] and probabilistic (i.e., relative-error) wavelet synopses [Garofalakis and Gibbons 2004]

for the *single-measure* case. We demonstrate that in the presence of multiple measures, choosing an effective subset of extended wavelet coefficients gives rise to difficult optimization problems which are significantly more complex than their single-measure counterparts. This is primarily due to our more involved extended-coefficient storage format that forces nontrivial dependencies between thresholding decisions made across different measures. We propose optimal solutions based on novel algorithmic formulations that employ dynamic programming (DP) ideas. Given the high time and space complexities of our exact DP schemes, we also introduce fast, greedy approximation algorithms (based on the idea of *marginal error-gains*) that produce near-optimal solutions. To the best of our knowledge, our work represents the first principled, methodical study of effective wavelet-based data reduction techniques for multimeasure datasets. More concretely, our key contributions can be summarized as follows:

—*Extended wavelet-coefficients for multimeasure data.* We provide a qualitative and quantitative demonstration of the suboptimal choices made by existing (Individual and Combined) strategies for the wavelet-based summarization of multimeasure datasets. Based on our observations, we formally define the notion of an extended wavelet coefficient, the first adaptive, efficient storage scheme for multimeasure wavelet-coefficients. Given a dataset comprised of M measures, an extended wavelet coefficient can be used to efficiently 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.

—*Extended wavelet-coefficient synopses for weighted sum-squared error.* We present a novel DP algorithm, termed DynProgL2, that selects the optimal subset of extended wavelet coefficients to retain in order to minimize the weighted sum of the squared L_2-error norms of all data measures, under a given storage constraint. The novelty of our DP formulation comes from the fact that the dependencies across measures due to our extended-coefficient storage format cause the key *principle of optimality* based on a *total ordering* of partial solutions [Cormen et al. 1990] to be violated, rendering straightforward (e.g., "knapsack-like") DP schemes inapplicable in our setting. Instead, our DynProgL2 algorithm relies on two mutually recursive DP recurrences that are tabulated in parallel to compute an optimal solution. We then introduce an alternative, greedy synopsis-construction algorithm for weighted L_2-error (termed GreedyL2) with significantly reduced memory and running-time requirements compared to the optimal DynProgL2 scheme. We also demonstrate that our GreedyL2 heuristic is *provably near optimal*, always guaranteeing a solution which is within a factor of, at most, $\min\{2, 1 + \frac{1}{\frac{B}{H+M}-1}\}$ of the optimal weighted L_2-error, for the given amount of space B and the maximum space $H + M$ that an extended wavelet coefficient may occupy.

—*Extended wavelet-coefficient synopses for relative error.* We address the problem of building probabilistic wavelet synopses (optimized for relative-error

metrics) [Garofalakis and Gibbons 2004] over multimeasure data. Once again, the cross-measure dependencies introduced by our space-efficient extended-coefficient storage format cause the principle of optimality to be violated, making the earlier single-measure DP solutions of Garofalakis and Gibbons [2004] inapplicable in the multimeasure context. Thus, we propose a novel probabilistic thresholding scheme for multimeasure datasets based on the idea of an exact *partial-order DP (PODP)* formulation. In a nutshell, our PODP solution (termed PODPRel) generalizes earlier single-measure DP schemes [Garofalakis and Gibbons 2004] to datasets with M measures by using an M*-component vector objective* and an M*-component less-than* partial order to prune subproblem solutions that cannot possibly be part of an optimal solution. Due to its more strict pruning criteria, PODPRel typically needs to tabulate and examine many more subproblem solutions, which can easily raise its time and space complexities to levels that are prohibitive in practice (e.g., time and space at least exponential in the number of measures involved).[1] Thus, we once again introduce a more efficient, greedy approximation algorithm (termed GreedyRel) for probabilistic coefficient thresholding over multimeasure data, which greedily allocates the available synopsis space based on marginal relative-error gains.

—*Extensive experimental results validating our approach.* We present results from an extensive experimental study of our proposed techniques with both synthetic and real-life datasets. Our results study the performance of our algorithms under a variety of different settings and parameters, and clearly verify the effectiveness of our approach. More specifically, our numbers demonstrate that our algorithms easily outperform existing approaches in terms of accuracy, often by an order of magnitude or more. Furthermore, our greedy approximation schemes always perform very close to optimal, thus providing effective and, at the same time, fast and scalable solutions to our multimeasure synopsis construction problems.

Due to space constraints, detailed proof arguments and several details of our technical development can be found in the Electronic Appendix that is accessible in the ACM Digital Library.

2. PRELIMINARIES

In this section, we provide a quick introduction to conventional Haar wavelet decomposition and wavelet-coefficient synopses in both one and multiple dimensions. We also discuss existing Individual and Combined strategies for handling multiple measures and demonstrate some of their important shortcomings. Finally, we introduce the notion of an extended wavelet coefficient which forms the basis for our proposed approach and data reduction algorithms.

[1]We also demonstrate that our PODP formulation can be appropriately extended (using ideas from the more recent work of Garofalakis and Kumar [2005]) to give optimal *deterministic* thresholding schemes for relative-error metrics. Unfortunately, in addition to the more strict, partial-order pruning, this extension also introduces an $O(N^M)$ factor in space/time complexity, rendering the approach unusable for any realistic problem sizes (Section 4.5).

2.1 One-Dimensional Haar Wavelets

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 [Stollnitz et al. 1996]. The wavelet decomposition has excellent energy compaction and decorrelation properties which can be used to effectively generate compact representations that exploit the structure of data. Suppose we are given 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 average values $[2, 1, 4, 4]$. In other words, the average of the first two values (i.e., 2 and 2) is 2, that of the next two values (i.e., 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 $2 - 2 = 0$, and 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 values of the original data array from the lower-resolution array containing the four averages and four detail coefficients. Recursively applying the aforementioned 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 *wavelet decomposition*) of A is that single coefficient representing the overall average of the data values, followed by the detail coefficients in 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 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 [Stollnitz et al. 1996].

Note that intuitively, wavelet coefficients carry different weights with respect to their importance in rebuilding the original data values. For example, the

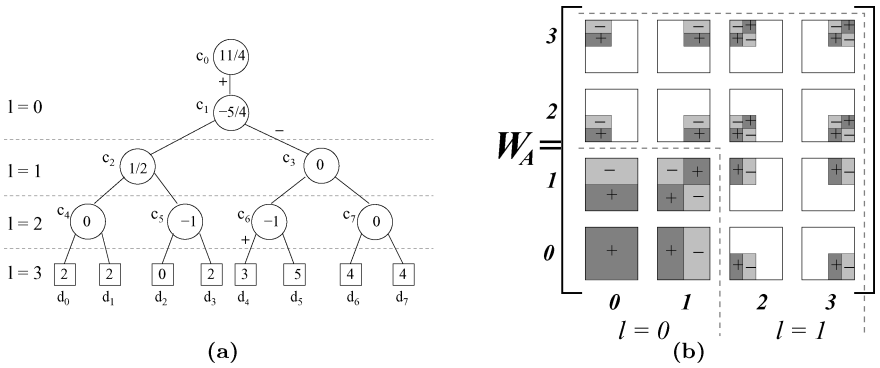

Fig. 1. (a) Error-tree structure for our example data vector A ($N = 8$); (b) support regions and signs for the 16 nonstandard two-dimensional Haar basis functions. The coefficient magnitudes are multiplied by $+1$ (-1) where a sign of $+$ (respectively, $-$) appears, and 0 in blank areas.

overall average is obviously more important than any detail coefficient, since it affects the reconstruction of all entries in the data array. In order to equalize the importance of all wavelet coefficients, we need to *normalize* the final entries of W_A appropriately. A common normalization scheme [Stollnitz et al. 1996] is to divide each wavelet coefficient by $\sqrt{2^l}$, where l denotes the *level of resolution* at which the coefficient appears (with $l = 0$ corresponding to the "coarsest" resolution level).

The Haar coefficient error tree. A helpful tool for exploring and understanding the key properties of Haar wavelet decomposition is the *error-tree* structure [Matias et al. 1998]. The error tree is a hierarchical structure built on the basis of the wavelet transform process (even though it is primarily used as a conceptual tool, an error tree can be easily constructed in linear $O(N)$ time). Figure 1(a) 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 positions in the data array or error tree. For example, c_0 corresponds to the overall average of A. The resolution levels l for coefficients (corresponding to levels in the tree) are also depicted (we use the terms "node" and "coefficient" interchangeably in the following). The normalized c_i coefficient is denoted by c_i^* ($= c_i/\sqrt{2^{\texttt{level}(c_i)}}$). Table I summarizes some of the key notational conventions used in this article (with obvious extensions to the multimeasure case); additional notation is introduced when necessary. Detailed symbol definitions are provided at appropriate locations in the text. For simplicity, the notation assumes one-dimensional wavelets; extensions to multidimensional wavelets (Section 2.2) are straightforward.

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 nonzero coefficients. A key property of Haar wavelet decomposition is that the reconstruction of any data value d_i depends only on those values of coefficients on $\texttt{path}(d_i)$; 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 $j = 0$,

Table I. Notation

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 dataset 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}, c^*_{ij}	Unnormalized/normalized Haar coefficient at coordinate i for the j^{th} measure
$\texttt{path}(u)$	All nonzero proper ancestors of u in the error tree
EC_i	Extended wavelet coefficient at coordinate i
H	Storage space for the extended coefficient header (coordinates and bitmap)

and $\delta_{ij} = -1$ otherwise. Thus, reconstructing any data value involves summing, at most, $\log N + 1$ coefficients. 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 for which c_i is used to reconstruct; the support region for a coefficient c_i is uniquely identified by its coordinate i.

2.2 Multidimensional Haar Wavelets

The Haar wavelet decomposition can be extended to *multidimensional* data arrays using two distinct methods, namely *standard* and *nonstandard* Haar decomposition [Stollnitz et al. 1996]. Each of these transforms results from a natural generalization of the one-dimensional decomposition process, and both have been used in a wide variety of applications, including approximate query answering over high-dimensional datasets [Chakrabarti et al. 2001; Vitter and Wang 1999].

The Haar decomposition of a D-dimensional data array A results in a D-dimensional wavelet-coefficient array W_A with the same dimension ranges and number of entries (the full details as well as efficient decomposition algorithms can be found in Chakrabarti et al. [2001], Vitter and Wang [1999]). Consider a D-dimensional wavelet coefficient W in the (standard or nonstandard) wavelet-coefficient array W_A. Here, W contributes to the reconstruction of a D-dimensional rectangular region of cells in the data array A (i.e., W's support region). Further, the sign of W's contribution ($+W$ or $-W$) can vary along the quadrants of its support region. As an example, Figure 1(b) depicts the support regions and signs of the 16 nonstandard, two-dimensional Haar coefficients in the corresponding locations of a 4×4 wavelet-coefficient array W_A. The blank areas for each coefficient correspond to those regions of A whose reconstruction is independent of the coefficient, that is, the coefficient's contribution is 0. Thus, $W_A[0, 0]$ is the overall average that contributes positively (i.e.,"$+W_A[0, 0]$") to the reconstruction of all values in A, whereas $W_A[3, 3]$ is a detail coefficient that contributes (with the signs shown in Figure 1(b)) only to values in A's upper-right quadrant. Each data cell in A can be accurately reconstructed by adding up the contributions (with the appropriate signs) of those

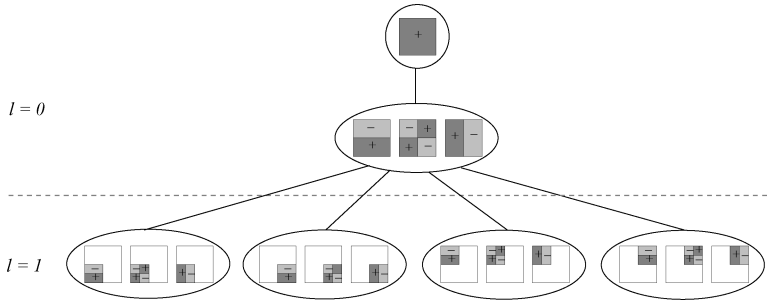

Fig. 2. Error-tree structure for the 16 nonstandard two-dimensional Haar coefficients for a 4×4 data array (data values omitted for clarity).

coefficients whose support regions include the cell. Figure 1(b) also depicts the two levels of resolution ($l = 0, 1$) for our example two-dimensional Haar coefficients; as in the one-dimensional case, these levels define the appropriate constants for normalizing coefficient values [Chakrabarti et al. 2001; Stollnitz et al. 1996].

Error-tree structures for multidimensional Haar wavelets can be constructed (once again, in linear $O(N)$ time) in a manner similar to those for the one-dimensional case, but their semantics and structure are somewhat more complex. A major difference is that in a D-dimensional error tree, each node (except for the root, i.e., the overall average) actually corresponds to a *set* of $2^D - 1$ wavelet coefficients that have the same support region, but different quadrant signs and magnitudes for their contribution. Furthermore, each (nonroot) node t in a D-dimensional error tree has 2^D children corresponding to quadrants of the (common) support region of all coefficients in t (note that the sign of each coefficient's contribution to the leaf (data) values residing at each of its children in the tree is determined by the coefficient's quadrant sign information).[2] As an example, Figure 2 depicts the error-tree structure for the two-dimensional 4×4 Haar coefficient array in Figure 1(b). Thus, the (single) child t of the root node contains the coefficients $W_A[0, 1]$, $W_A[1, 0]$ and $W_A[1, 1]$ and has four children corresponding to the four 2×2 quadrants of the array; the child corresponding to the lower-left quadrant contains the coefficients $W_A[0, 2]$, $W_A[2, 0]$ and $W_A[2, 2]$, and all coefficients in t contribute with a "+" sign to all values in this quadrant.

Based on the preceding generalization of the error-tree structure to multiple dimensions, we can naturally extend the process for data value reconstruction to multidimensional Haar wavelets. Once again, the reconstruction of d_i depends only on the *coefficient sets* for all error-tree nodes in $\texttt{path}(d_i)$, where the sign of the contribution for each coefficient W in node $t \in \texttt{path}(d_i)$ is determined by the quadrant sign information for W.

[2]The number of children (coefficients) for an internal error-tree node can actually be less than 2^D (respectively, $2^D - 1$) when the sizes of data dimensions are not all equal. In these situations, the exponent for 2 is determined by the number of dimensions that are *active* at the current level of the decomposition (i.e., those dimensions that are still being recursively split by averaging/differencing).

2.3 Wavelet-Based Data Reduction: Coefficient Thresholding

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 in W_A as a highly compressed approximate representation of the original data (the remaining coefficients are implicitly set to 0). The goal of coefficient thresholding is to determine the "best" subset of B coefficients to retain so that some overall error measure in the approximation is minimized. The method of choice for the vast majority of earlier studies on wavelet-based data reduction and approximation [Chakrabarti et al. 2001; Matias et al. 1998; Matias et al. 2000; Vitter and Wang 1999] is *conventional coefficient thresholding* that greedily retains the B largest Haar wavelet coefficients in *absolute normalized value*. This thresholding method is in fact *provably optimal* with respect to minimizing the overall sum-squared error (i.e., L_2-norm error) in the data compression [Stollnitz et al. 1996].

More formally, letting $\hat{d}_i$ denote the (approximate) reconstructed data value for cell i, retaining the B largest normalized coefficients implies that the resulting synopsis minimizes the quantity $\sum_i (\hat{d}_i - d_i)^2$ (for the given amount of space B). This fact follows from the *orthonormality* of the normalized Haar wavelet basis which, by *Parseval's theorem*, implies that the energy of the signal is the same in both the data and the (normalized) wavelet domain; that is, $\sum_i d_i^2 = \sum_i (c_i^*)^2$ (thus, the "energy" loss is minimized when dropping the smallest normalized coefficients from the synopsis).

Each Haar coefficient is stored in the wavelet synopsis as a pair $< i, c_i^* >$, where i denotes the index/coordinate of the coefficient and c_i^* denotes its (normalized) value. In the case of D-dimensional data, each (nonzero) synopsis coefficient is stored as a $(D+1)$-tuple $< i_1, i_2, \ldots, i_D, c^*_{i_1,i_2,\ldots,i_D} >$, where $i_1, \ldots, i_D$ denote the coefficient's coordinates in the (D-dimensional) wavelet-coefficient array W_A.

2.4 Existing Approaches for Multiple Measures

Two existing approaches [Stollnitz et al. 1996] (termed Individual and Combined) have been proposed for adapting wavelet-based data reduction to datasets with multiple measures—both are straightforward generalizations of the single-measure case. The Individual strategy performs an independent wavelet decomposition for each individual measure, and the decisions on which coefficients to retain are made independently for each measure. In the Combined approach, both the original data values and the produced wavelet coefficients are treated as M-component vectors (where M denotes the number of data measures). The pairwise averaging and differencing procedure described before is then performed between corresponding vector components (i.e., values for the same measure). The Combined thresholding procedure is very similar to that for the single-measure case: The coefficient vectors retained in the synopsis are those with the largest values for the L_2-vector norm.

We also use the terms *individual* and *combined* coefficient to refer to the coefficient values that result from the corresponding decomposition algorithms. Thus, a combined coefficient is an M-component vector that stores individual

Table II. Example Combined Wavelet Decomposition

Resolution	Averages	Detail Coefficients
3	$\left[\begin{bmatrix}2\\4\end{bmatrix}\begin{bmatrix}2\\6\end{bmatrix}\begin{bmatrix}0\\3\end{bmatrix}\begin{bmatrix}2\\5\end{bmatrix}\begin{bmatrix}3\\2\end{bmatrix}\begin{bmatrix}5\\8\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}\right]$	—
2	$\left[\begin{bmatrix}2\\5\end{bmatrix}\begin{bmatrix}1\\4\end{bmatrix}\begin{bmatrix}4\\5\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}\right]$	$\left[\begin{bmatrix}0\\-1\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix}\begin{bmatrix}-1\\-3\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}\right]$
1	$\left[\begin{bmatrix}3/2\\9/2\end{bmatrix}\begin{bmatrix}4\\4\end{bmatrix}\right]$	$\left[\begin{bmatrix}1/2\\1/2\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\right]$
0	$\left[\begin{bmatrix}11/4\\17/4\end{bmatrix}\right]$	$\left[\begin{bmatrix}-5/4\\1/4\end{bmatrix}\right]$

coefficient values for each of the M measures in the dataset (at given coordinates). An example of the Combined decomposition algorithm is shown in Table II. Our example dataset here is comprised two measures: The values for the first measure are identical to those in our first example array in Section 2.1, while the values for the second are [4, 6, 3, 5, 2, 8, 3, 3]. Thus, in Table II, the first (second) row of each vector corresponds either to data or coefficient values for the first (respectively, second) measure. The final set of combined coefficients is $W_A = \left[\begin{bmatrix}11/4\\17/4\end{bmatrix}, \begin{bmatrix}-5/4\\1/4\end{bmatrix}, \begin{bmatrix}1/2\\1/2\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0\\-1\end{bmatrix}, \begin{bmatrix}-1\\-1\end{bmatrix}, \begin{bmatrix}-1\\-3\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}\right]$.

The Combined data reduction strategy is expected to achieve better storage utilization than the Individual algorithm for the L_2-error metric in datasets where multiple component values for the same combined coefficient are simultaneously "large" (in terms of their absolute normalized value). In such cases, the coordinates of such large combined coefficients are stored only once, thus allowing for a more compact representation. More compact representations imply the ability to store larger numbers of coefficient values and, thus, improved result accuracy (for a given space budget). On the other hand, in many scenarios, a combined coefficient might help to reduce the error significantly in only one or few measures. In such cases, some of the space occupied by the combined-coefficient components is essentially wasted, without improving the overall quality of the approximate results.

Table III depicts only two combined coefficients for a one-dimensional dataset with three measures. Neither the actual data that helped construct these two coefficients nor the remaining set of coefficients are important, since the sole purpose of this example is to show that both Individual and Combined strategies can result in poor choices, even when choosing between just two coefficients. We assume that each dimension coordinate and each coefficient value require one unit of space. In this scenario, each combined-coefficient tuple occupies four space units (one coordinate + three measure values), while each individual coefficient occupies only two space units. For a storage constraint of four units of space, the Combined algorithm can thus select only one tuple to store, while the Individual algorithm can store up to two individual coefficients. By Parseval's theorem (Section 2.3), the benefit of retaining any single coefficient value is equal to its squared normalized value. As Table III shows, in Case A, for the given storage constraint, the Combined algorithm chooses a solution with only

Table III. Shortcomings of the Combined and Individual Strategies

Case A					Case B				
	Coordinate	Values				Coordinate	Values		
Available Coefficients	0	100	0	0	Available Coefficients	0	100	100	100
	1	0	100	0		1	0	100	0
	Coordinate	Values				Coordinate	Values		
Combined Retains	0	100	0	0	Combined Retains	0	100	100	100
	Coordinate	Value	Measure			Coordinate	Value	Measure	
Individual Retains	0	100	1		Individual Retains	0	100	3	
	1	100	2			0	100	2	
Combined Benefit = $100^2 = 10000$					Combined Benefit = $100^2 + 100^2 + 100^2 = 30000$				
Individual Benefit = $100^2 + 100^2 = 20000$					Individual Benefit = $100^2 + 100^2 = 20000$				
$\frac{\text{Combined Benefit}}{\text{Individual Benefit}} = \frac{10000}{20000} = 50\%$					$\frac{\text{Individual Benefit}}{\text{Combined Benefit}} = \frac{20000}{30000} \approx 66.7\%$				

half the benefit of the solution picked by the Individual algorithm. The roles are reversed in Case B, where the Individual algorithm selects a solution with only two-thirds of the benefit achieved by the Combined algorithm. Note that in Case B, the ties on retained coefficients for the Individual algorithm are broken arbitrarily, as four individual coefficients have the same benefit. By increasing the number of measures in the dataset for Case A and the number of dimensions for Case B, we can easily create examples where the quality of the suboptimal solutions returned by the Combined and Individual strategies (respectively) is significantly worse than the optimal choice. For instance, by expanding our one-dimensional dataset of Table III to a dataset with M measures, and considering which $\frac{M+1}{2}$ candidate individual coefficients to retain under a storage constraint of $M+1$ space units, it can be shown that in Case A: $\frac{\text{Combined Benefit}}{\text{Individual Benefit}} = \frac{2}{M+1}$, while in Case B: $\frac{\text{Individual Benefit}}{\text{Combined Benefit}} = \frac{M+1}{2M}$.

An additional disadvantage of the Combined data reduction strategy is that it cannot be easily adapted for cases where we would like to assign different weights on the quality of answers for different measures. For example, in colored image databases, datasets form two-dimensional arrays with three measures, namely, the pixel values for each of the three basic colors (red, green and blue). It has been shown [Foley et al. 1990] that better image compression is possible by first converting the image values from the RGB to the YIQ color space, thus separating the luminance (Y) from the chromatic information (I and Q). Since human perception is more sensitive to variations in Y and less to those in Q, we may want to specify a larger weight for errors in Y and a smaller one for errors in Q. In such scenarios, use of the Combined algorithm becomes problematic, since it cannot devote different fractions of the available space to different measures, even though the coefficient values within each vector are weighted differently. Moreover, for datasets with several measures, it seems quite unlikely that the coefficient values across *all* measures would be simultaneously large or small (i.e., all measure values in the data are positively correlated). For such datasets, the Combined algorithm would clearly waste synopsis storage space, without significantly improving the overall accuracy of the approximation.

On the other hand, there are cases when we can expect multiple coefficient values of a combined coefficient to have large values. As discussed in Sections 2.1–2.2, Haar coefficient values are normalized based on their respective resolution levels. Due to this normalization, coefficient values at lower (i.e., "coarser") resolution levels typically tend to have larger values, and this occurs for all measures. Another scenario where multiple large coefficient values might occur at the same coordinate(s) arises for *sparse* datasets (typical in real-life high-dimensional data analysis applications). In such datasets, there often are sparse regions of the data space with only one or very few data tuples present. Depending on the sizes of such regions and the tuple data values, such "spikes" in the input data signal can potentially create large Haar coefficient values across many measures. Clearly, in such scenarios, the Combined algorithm can provide significant advantages over an Individual strategy by avoiding the replication of coordinates for multiple measures in the synopsis.

2.5 Our Approach: Extended Wavelet Coefficients

The aforementioned Individual and Combined schemes essentially represent the two extremes of the design spectrum by assuming that either *one* or *all* values of a Haar coefficient are important for the synopsis (and can share their coordinates). Given the important shortcomings of both techniques when dealing with multimeasure datasets, we now introduce the notion of an extended wavelet coefficient that tries to bridge the gap between these two extremes by providing an efficient, flexible storage format for retaining *any subset* of coefficient values.

Definition 1. An extended wavelet coefficient EC for a D-dimensional dataset with M measures is defined as a triple $EC = \langle C, \beta, V \rangle$ 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.

The bitmap of an extended wavelet coefficient determines exactly which of the (at most M) per-measure values of the combined coefficient at the given coordinates C have actually been stored. Thus, an extended wavelet coefficient combines the positive aspects of both the Individual and Combined algorithms as a flexible storage method that can store anywhere from one to M values for any combination of coefficient coordinates (we refer to the (coordinates, bitmap) pair for an extended wavelet coefficient as the coefficient's *header*).

In the remainder of the article, we address the problem of building effective extended wavelet-coefficient synopses (under a given storage constraint) for different classes of target error metrics in the approximate data reconstruction. We use $|EC|$ to denote the space requirement of an extended coefficient EC. Of course, this space requirement varies, depending on the actual contents (i.e., number of stored coefficient values) of EC. Since our focus is on selecting the specific coefficient values to store, our algorithms address only the final *thresholding* step of the wavelet-based data reduction process. The input to all of our thresholding schemes is the complete set of combined wavelet coefficients,

which can be trivially created using the same decomposition process as in either the Individual or Combined algorithm (Section 2.4).

3. EXTENDED WAVELET SYNOPSES FOR WEIGHTED SUM-SQUARED ERROR

As discussed earlier, the most common optimization objective for conventional wavelet synopses in the single-measure case is the sum-squared (i.e., L_2) error in the data approximation. Thus, a natural extension for datasets with multiple measures is to optimize for a weighted sum-squared error across all data measures. More formally, our optimization problem can be stated as follows.

Weighted Sum-Squared Error Minimization for Extended Coefficients. Given a collection W_A of candidate combined wavelet coefficients of a D-dimensional dataset with M measures, a storage constraint B, and an M-vector of measure weights $\bar{w}$, select a synopsis $\mathcal{S}$ of extended wavelet coefficients that minimizes the weighted sum of the L_2-error norms across all measures; that is, minimize $\sum_{j=1}^{M} \left(w_j \times \sum_i (d_{ij} - \hat{d}_{ij})^2\right)$ subject to the constraint $\sum_{EC \in \mathcal{S}} |EC| \leq B$.

Based on Parseval's theorem and the discussion in Section 2.3, and using c_{ij}^* to denote the normalized value for the j^{th} measure of the i^{th} input combined wavelet coefficient, we can restate the previous optimization problem in the following equivalent (and easier to process) form.

Weighted Sum-Squared Benefit Maximization for Extended Coefficients. Given a collection W_A of candidate combined wavelet coefficients of a D-dimensional dataset with M measures, a storage constraint B, and an M-vector of measure weights $\bar{w}$, select a synopsis $\mathcal{S}$ of extended wavelet coefficients that maximizes the weighted sum of the retained normalized squared coefficient values across all measures; that is, maximize $\sum_{EC=\langle C,\beta,V\rangle \in \mathcal{S}} \sum_{j,\beta(j)=1} w_j \times (c_{ij}^*)^2$ subject to the constraint $\sum_{EC \in \mathcal{S}} |EC| \leq B$.

3.1 DynProgL2: An Optimal Dynamic Programming Algorithm

We now propose a thresholding algorithm (termed DynProgL2), based on dynamic programming (DP) ideas, that optimally solves the optimization problem described previously. Our DynProgL2 algorithm takes as input a set of combined coefficients W_A, a space constraint B, and an M-vector of weights $\bar{w}$ (used to weight the benefit of the coefficient values for each measure). DynProgL2 treats the individual coefficient values for each input combined coefficient as *subitems* in our problem, utilizing an implicit mapping that maps the j^{th} coefficient value of the i^{th} combined coefficient to the subitem index $k = (i-1) * M + j$ (thus, the M per-measure values for each combined coefficient correspond to consecutive subitems). Our discussion in this section makes use of this mapping so as to simplify the development of our algorithms. Finally, we should emphasize here that our DynProgL2 thresholding algorithm is applicable (without any modifications) independent of data dimensionality, since increasing the dimensionality only affects the header size H for the retained extended wavelet coefficients.

Let $k = (i-1) * M + j$ denote the subitem index corresponding to the j^{th} coefficient value of the i^{th} combined coefficient. Retaining the k^{th} subitem in our synopsis gives us a weighted benefit equal to $w_j \times (c_{ij}^*)^2$. However, the space

overhead for storing this subitem obviously depends on whether this is the *first* coefficient value being stored from the i^{th} combined coefficient. If this is indeed the first value retained from the i^{th} combined coefficient, then its space requirements are equal to the space needed to store the extended coefficient header (coordinates and bitmap) plus the space for storing the coefficient value. On the other hand, if other subitems for the coefficient have already been stored, then we have already paid the space penalty for the coefficient header, and the subitem's space requirements are simply the space for storing the coefficient value. These dependencies on the storage-space requirements across coefficient values (due to the shared space for extended coefficient headers) render the design of an optimal solution to our problem significantly more complex than that of known (pseudopolynomial) DP algorithms to traditional *Knapsack-style* problems [Vazirani 2001] (note, of course, that in our problem scenario, the space bound B is always upper bounded by $|W_A|$ and, thus, is polynomial (linear) in the input size).

We now try to formulate a DP recurrence for our optimization problem. Note that in order to be able to tabulate partial solutions in our DP table, our development here requires that all subitems occupy an integer number of space units—this can be done, for instance, by assuming a basic space unit of one bit.[3] We use `sizeof(float)` to denote the number of space units needed to store a single (floating-point) coefficient value. For the optimal solution using a synopsis space of (at most) S, and considering the first k subitems, three cases may arise:

(1) The optimal solution is the same as that using the first $k-1$ subitems and the same space S;
(2) the optimal solution is achieved by including subitem k, and k is the *first* subitem of its combined coefficient included in the optimal solution; or,
(3) the optimal solution is achieved by including subitem k, and k is not the first subitem of its combined coefficient included in the optimal solution.

It is important to note that in the third case, the k^{th} subitem needs to be combined with the optimal partial solution P that: (a) uses (at most) the first $k-1$ subitems and space of at most $S -$ `sizeof(float)`; *and* (b) includes *at least one more* subitem (i.e., other than k) from the corresponding i^{th} combined coefficient. This second requirement results in what may be surprising observation: The partial solution P is *not necessarily optimal* for our optimization problem when using only the first $k-1$ subitems and up to $S -$ `sizeof(float)` units of space. An example is shown in Table IV, which depicts just two coefficients corresponding to a three-dimensional dataset with three measures. To keep things simple, assume that the storage bound B is equal to the size of one tuple augmented by a bitmap of three bits, and that each measure has a weight

[3]Some techniques, like encoding the bitmap within some coefficient coordinate(s) for small numbers of measures, can help to increase the size of the space unit, thus decreasing the memory requirements of our DP tables. Still, such optimizations do not improve the asymptotic space complexity of our problem.

Table IV. Unexpected Optimal Solution Arises for Space Bound $S_1 + \texttt{sizeof(float)}$

Candidate Coefficients					
Coordinates			Values		
0	0	1	100	1	2
1	2	0	99	98	97

Considered Subitems	SubItems in Optimal Solution For Space Bound		
	S_1	$S_1 + \texttt{sizeof(float)}$	$S_1 + 2\texttt{sizeof(float)}$
First 1	1	1	1
First 2	1	1,2	1,2
First 3	1	1,3	1,2,3
First 4	1	1,3	1,2,3
First 5	1	**4,5**	4,5
First 6	1	4,5	4,5,6

of 1. Note that under this storage bound, it is obviously impossible to store two coefficient values from different coefficients. Let S_1 denote the space needed to store a single coefficient value, along with the pertinent header information; that is, $S_1 = H + \texttt{sizeof(float)}$. Now, consider the optimal solution for space bound $S_1 + \texttt{sizeof(float)}$, when considering up to the first five subitems. It is easy to see that at this point, the optimal solution is to store subitems 4 and 5 (both corresponding to the second combined coefficient); however, it is also easy to see that subitem 4 is not part of *any* optimal solution involving only the first four subitems (for any storage bound $\leq B$), since subitem 1 can always be used in its place to give a solution with larger benefit.

The preceding discussion basically shows that our optimization problem for extended wavelet synopses basically violates the key principle of optimality for conventional dynamic programming [Cormen et al. 1990], and requires us to come up with a novel algorithmic solution. An important observation here is that we need only store, for each possible subitem$\times$space (k, S) combination, a single suboptimal solution, namely, the best solution (for at most S units of space and considering up to the k^{th} subitem) which *forces* at least one subitem of the combined coefficient corresponding to k to be included in the synopsis. Our dynamic program employs an array FORCE$[k, S]$ to tabulate such suboptimal solutions (for each subitem$\times$space combination), in addition to the more conventional OPT$[k, S]$ DP array which tabulates the (partial) optimal solutions to our problem (when using space $\leq S$ and considering the first k subitems).

Our DynProgL2 algorithm (depicted in Figure 3) computes entries for the OPT$[k, S]$ and FORCE$[k, S]$ arrays (both of size $(N \cdot M) \times B$) in a mutually recursive manner. Each cell of these two arrays is comprised of two numeric fields: (1) a "benefit" field recording the total benefit for the corresponding partial solution, and (2) a "choice" field used to code the choice made by our dynamic program when deciding the benefit of a cell, to be explained shortly (both fields are necessary in order to retrace the actions of our algorithm when building the optimal solution). DynProgL2 begins by initializing some entries for both the OPT and FORCE arrays: Lines 1–3 are based on the fact that no coefficient value can be stored in space less than $H + \texttt{sizeof(float)}$. Similarly, the optimal solution for space of at least $H + \texttt{sizeof(float)}$ and considering only the first subitem obviously only includes this subitem (Lines 4–6). DynProgL2 then iteratively fills in the values for the remaining cells (Lines 7–17). For the OPT array, the benefit for the optimal solution, using space $\leq S$ and considering up

procedure DynProgL2(W_A, B, $\overline{w}$)
Input: $N \times M$ vector of combined wavelet coefficients W_A; space constraint B; per-measure weight vector $\overline{w}$.
Output: Optimal set of extended wavelet coefficients and benefit of optimal solution.

1. **for each** $S < H +$ `sizeof(float)` **do** // initialize DP array entries
2. OPT$[*, S]$.benefit := FORCE$[*, S]$.benefit := 0
3. OPT$[*, S]$.choice := FORCE$[*, S]$.choice := 1
4. **for each** $S \geq H +$ `sizeof(float)` **do**
5. OPT$[1, S]$.benefit := FORCE$[1, S]$.benefit := $w_1 \times (c^*_{1,1})^2$
6. OPT$[1, S]$.choice := FORCE$[1, S]$.choice := 3
7. **for** $k := 2$ **to** $N \cdot M$ **do**
8. let $i := 1 + (k-1) \div M$ // combined-coefficient index
9. let $j := 1 + (k-1) \mod M$ // measure index
10. let $f :=$ `sizeof(float)` // space for a single coefficient value
11. **for** $S := H + f$ **to** B **do**
12. OPT$[k, S]$.benefit := $\max \begin{cases} \text{OPT}[k-1, S].\text{benefit} \\ \text{OPT}[k-1, S-H-f].\text{benefit} + w_j \times (c^*_{i,j})^2 \\ \text{FORCE}[k-1, S-f].\text{benefit} + w_j \times (c^*_{i,j})^2 & j > 1 \end{cases}$
13. Depending on which of the three choices listed above produced the maximum-benefit value, set OPT$[k, S]$.choice equal to 2, 3,or 4 (respectively)
14. FORCE$[k, S]$.benefit := $\max \begin{cases} \text{FORCE}[k-1, S].\text{benefit} & j > 1 \\ \text{OPT}[k-1, S-H-f].\text{benefit} + w_j \times (c^*_{i,j})^2 \\ \text{FORCE}[k-1, S-f].\text{benefit} + w_j \times (c^*_{i,j})^2 & j > 1 \end{cases}$
15. Depending on which of the three choices listed above produced the maximum-benefit value, set FORCE$[k, S]$.choice equal to 2, 3,or 4 (respectively)
16. **endfor**
17. **endfor**
18. Build the optimal solution by doing a reverse traversal starting from the entry OPT$[N \cdot M, B]$, and moving based on the choice field of the current entry
19. **return**(OPT$[N \cdot M, B]$.benefit) // return benefit of optimal synopsis

end

Fig. 3. The optimal DynProgL2 algorithm.

to the first k subitems, is computed as the best (i.e., maximum-benefit) choice from the three cases described before (Lines 12–13). A similar choice is made for the corresponding best solution for the FORCE array entries (Lines 14–15). Note that some of the three cases are valid only if the current subitem satisfies some conditions, namely, that it corresponds to a coefficient value with a measure index of at least two (i.e., $j > 1$). The "choice" field of our DP array entries, which codes the choice made when determining the benefit of an entry, is assigned a value of 2, 3, or 4 (respectively), depending on which of the three aforementioned cases produced the optimal solution. Entries corresponding to cases where no subitem can be stored in the specified space have a "choice" field value of 1 (Line 3).

At the end, the total benefit for the optimal extended wavelet synopsis is that achieved when considering all $N \cdot M$ subitems and using (at most) B space units, namely, OPT$[N \cdot M, B]$ benefit. We can build the optimal solution by retracing

the actions of DynProgL2, starting from cell $[N \cdot M, B]$ and moving on the basis of the "choice" field of the current cell. More formally, our optimal synopsis construction process starts by initializing the synopsis $\mathcal{S}$ to ϕ and, assuming that (at some point) we are at cell $[k, S]$, the action performed is determined based on the value of this cell's "choice" field, as follows: (1) choice $= 1$: end of traversal; (2) choice $= 2$: move to cell $[k-1, S]$ of the same array; (3) choice $= 3$: add an extended wavelet coefficient containing the k^{th} subitem to $\mathcal{S}$, and move to cell $[k-1, S-H-$ `sizeof(float)`$]$ of the OPT array; and (4) choice $= 4$: add the k^{th} subitem to $\mathcal{S}$ (creating a new extended wavelet coefficient, if necessary), and move to cell $[k-1, S-$ `sizeof(float)`$]$ of the FORCE array.

Time and space complexity. The space requirements of our DynProgL2 algorithm are essentially determined by the size of the OPT and FORCE arrays, which is $O(NMB)$. Given that the value of each cell is computed in constant $O(1)$ time for both of our DP arrays, the overall time complexity of DynProgL2 is also $O(NMB)$. Our reverse-traversal procedure for constructing the optimal extended wavelet synopsis takes $O(NM)$ time, since each step essentially checks (in constant time) whether a subitem k belongs in the synopsis, and then proceeds to subitem $k-1$.

3.2 GreedyL2: An Efficient, Provably Near-Optimal Approximation Algorithm

We now present a greedy solution to the optimization problem of Section 3. Our algorithm, termed GreedyL2, is based on transforming our optimization problem to a variant of the 0-1 Knapsack problem, and then selecting which coefficient values to store based on a *per-space benefit* metric. To simplify the exposition, our development here assumes that the unit of storage space is equal to the space needed to store a single coefficient value (i.e., `sizeof(float)`). Once again, note that our GreedyL2 algorithm can be applied without any modifications, independent of data dimensionality, since the increased dimensionality simply affects the size of the header of any stored extended wavelet coefficients.

Similar to the dynamic programming algorithm presented in the previous section, GreedyL2 receives as input a set of candidate combined wavelet coefficients W_A, a set of weights $\overline{w}$, and a storage constraint B. Instead of considering the benefit of each coefficient value individually, GreedyL2 considers at each step the optimal benefit achieved by selecting a subset of k $(1 \leq k \leq M)$ coefficient values (of the same combined coefficient) that have not already been stored. It easy to see that the optimal selection includes nonstored coefficient values corresponding to the k largest benefits: $w_j \times (c_{ij}^*)^2$, where w_j is the measure weight corresponding to the coefficient, and c_{ij}^* is the normalized coefficient value. The storage space for these k values will be equal to $H+k$ if no value of this combined coefficient has been stored before, and k otherwise. GreedyL2 maintains a structure with all optimal sets of size k $(1 \leq k \leq M)$ of all combined coefficients, and selects that set with the largest per-space benefit. The coefficient values belonging to the selected set are stored, and the benefits of optimal sets for the chosen combined coefficient have to be recalculated to only consider values that have not already been stored.

procedure GreedyL2(W_A, B, W)
Input: $N \times M$ vector of combined wavelet coefficients W_A; space constraint B;
per-measure weight vector $\overline{w}$.
Output: Selected set of extended wavelet coefficients and benefit of greedy solution
1. A max-heap structure *struct* is used to maintain the optimal benefits of the candidate sets of coefficient values.
2. Each entry in *struct* has 4 fields: (1) *psb:* per space benefit, (2) *num:* number of coefficient values in candidate set, (3) *space:* space needed for storing the set's coefficient values, and (4) *index:* index of combined coefficient the set belongs to.
3. $Stored[i]$ denotes the number of coefficient values from the ith input combined coefficient that have already been selected to be stored.
4. **for** $i := 1$ **to** N **do**
5. Sort coefficient values in descending order of their weighted benefit $w_j \times (c^*_{ij})^2$
6. Set $SortOrder[i, j]$ to the index of the measure with the jth top weighted benefit
7. $Stored[i] := 0$
8. InsertSets($i, struct, Stored, SortOrder, \overline{w}$)
9. **endfor**
10. SpaceLeft := B
11. **while** ($SpaceLeft \geq 1$) **and** ($struct.size() > 0$) **do**
12. **repeat** $PickedSet = struct.pop()$ **until** $PickedSet.space \leq SpaceLeft$
13. Also remove all candidate sets of the combined coefficient $PickedSet.index$
14. SpaceLeft -= $PickedSet.space$
15. $Stored[PickedSet.index] += PickedSet.num$
16. Remove from *struct* all sets belonging to coefficient $PickedSet.index$
17. InsertSets($PickedSet.index, struct, Stored, SortOrder, \overline{w}$)
18. **endwhile**
19. For each combined coefficient store the Stored coefficient values with the largest weighted benefit
end

procedure InsertSets(i, *struct*, *Stored*, *SortOrder*, $\overline{w}$, *SpaceLeft*)
Input: Index i of input combined coefficient; structure *struct* of candidate sets;
Symbols *Stored*, *SortOrder*, $\overline{w}$ and *SpaceLeft* defined as in GreedyL2 algorithm
Output: Candidate sets inserted in *struct* structure
begin
1. $cumulativeBen := 0$
2. **for** $j := Stored[i] + 1$ **to** M **do**
3. p = $SortOrder[i, j]$
4. $cumulativeBen += w_p \times (c^*_{ip})^2$
5. **if** ($Stored[i] > 0$) **then** $spaceNeeded = j - Stored[i]$
6. **else** $spaceNeeded = H + j$
7. **if** ($spaceNeeded < SpaceLeft$) **and** ($(c^*_{ip})^2 > 0$) **then**
8. struct.insert(candidateSet($\frac{cumulativeBen}{spaceNeeded}$, j, $spaceNeeded$, i))
9. Also connect all inserted sets of same combined coefficient using a cyclic list
10. **endfor**
end

Fig. 4. The GreedyL2 algorithm.

More formally, for each input combined coefficient, our GreedyL2 algorithm (depicted in Figure 4) first decides the sort order of its coefficient values based on their weighted benefit (Lines 5–6). For each combined coefficient, we also maintain the number of its coefficient values that have been selected for storage in a *Stored* array, whose entries are initialized to zero at the beginning of the algorithm (Line 7). Due to the way our algorithm is formulated, we do not need

to remember which of the coefficient i's values have been selected for storage, since these will always be the ones with the $Stored[i]$ maximum weighted benefits. We then calculate the optimal benefits of sets containing k coefficient values, $1 \leq k \leq M$ (Line 8). The maximum number of such sets is, at most M, but not always equal to M, since we do not need to create any sets that include any coefficient values with zero benefit. The space needed to store each of these k sets is $H+k$. The per-space benefit of each set, along with its occupied space, the number of coefficient values within this set, and the identifier of the coefficient that it belongs to, are then inserted in a max-heap structure,[4] where its elements are ordered based on their per-space benefit. We chose to use such a structure since each of the insert, delete, and finding-the-maximum-value operations has logarithmic cost. However, any other data structure with similar characteristics can be used in its place.

The algorithm then repeatedly (Lines 11–18) picks that set with the maximum per-space benefit which can fit in the remaining space. The values corresponding to this set are uniquely identified by the identifier field of the corresponding combined coefficient (stored in the *index* field of each set), its *Stored* variable, and the size of the chosen set. For the corresponding combined coefficient of the selected set, the optimal benefits of its sets have to be recalculated to include only nonstored coefficient values. This coefficient's previous sets are then removed from the tree and the newly calculated ones inserted. Note that the space required for newly inserted sets does not include the size of the header, since this has already been taken into account. The entire procedure terminates when no set can be stored without violating the storage constraint ($SpaceLeft < 1$). In order to create the output extended coefficients, we simply have to parse the list of combined coefficients, and for any coefficient that has a *Stored*[] value greater than 0, create an extended wavelet coefficient and store in this its *Stored*[] coefficient values with the largest weighted benefits. The following theorem bounds the worst-case performance of our algorithm.

THEOREM 1. *The* GreedyL2 *algorithm has a guaranteed approximation ratio bound of* $\min\{2, 1 + \frac{1}{\frac{B}{H+M}-1}\}$.

Time and space complexity. Each of the N input combined coefficients creates at most M candidate sets. Therefore, the space for the max-heap is $O(NM)$. For each combined coefficient, maintaining the sort order requires $O(M)$ space. The size of the input combined coefficients is $O(N(D+M))$, making the overall space complexity of the algorithm $O(N(D+M))$.

Determining the sort order for the values of each combined coefficient requires time $O(M \log M)$. Calculating the benefits of the sets produced by each coefficient then takes only $O(M)$ time. The original construction of a max-heap with $O(NM)$ elements can be done in $O(NM)$ time. Thus, the overall running

[4]The use of a max-heap structure, in contrast to the proposed AVL-tree used in the Deligiannakis and Roussopoulos [2003a] paper, helps to trim a logarithmic factor from the initial construction time of the algorithm. A min-heap structure was used by the authors of Guha et al. [2004] in an improved version of this algorithm for streaming environments.

time for the max-heap construction is $O(NM \log M)$. Each time a set is chosen for inclusion in the result, the search requires $O(\log(NM))$ time. Thus, we need to make $O(M)$ deletions from the max-heap, corresponding to all sets of the chosen combined coefficient. Finding all such nodes on the tree requires $O(M)$ time if they are connected by a cyclic list. Note that all the sets of the same combined coefficient are created at the same time, thus making it easy to create such a list. Each of the $O(M)$ insertion and deletion operations then requires $O(\log(NM))$ time. Since, at most, $O(\frac{M \times B}{H+M})$ sets can be selected (for each extended wavelet coefficient, the smallest average space per stored coefficient value occurs when all the M coefficient values are stored), the total time complexity should be $O(NM \log M + \frac{BM}{H+M} \times M \log(NM))$. However, a small complication arises because the algorithm may at some point repeatedly select candidate sets that do not fit within the remaining space. Since at most $O(NM)$ such sets may be selected and removed, the running time cost of this step might dominate the algorithm's running time. So, in this case, we allow the algorithm to deviate from its behavior of removing that candidate set with the maximum per-space benefit and instead scan the entire heap for that set with the maximum per-space benefit *that fits within the remaining space SpaceLeft*. Since the maximum value of the remaining space when this occurs must be less than $H + M$, and the minimum size of each candidate set is 1, at most, $O(H + M)$ steps with linear search cost may be incurred. This results in a total running time complexity of $O(NM(H + M) + \frac{BM}{H+M} \times M \log(NM))$.

Since the selection order of candidate sets is based on their per-space benefit, for each combined coefficient, only that candidate set with the maximum per-space benefit can be selected at each step. The only exception may occur at the finalization phase of the algorithm, namely, if this candidate set does not fit within the remaining synopsis space. By our analysis (Appendix A and Deligiannakis and Roussopoulos [2003b]), the solution tracked by GreedyL2 is, in fact, *optimal* when this first occurs for any selected candidate set. To further improve the running time of the algorithm, we may ignore the small unused space and thus, always only insert (at most) one candidate set for each combined coefficient, namely, that set with the maximum per-space benefit, since this is the only one that may be selected for inclusion in the final solution. This reduces the space of the max-heap to $O(N)$ and requires $O(1)$ deletion and insertion operations for each chosen set, while maintaining the same tight approximation ratio bound. Thus, we get an improved running time of $O(NM \log M + \frac{BM}{H+M} \log N)$. In subsequent work, Guha et al. [2004] give an improved version of this algorithm which: (1) further reduces the space requirements to $O(B)$, and (2) can work on streaming data (a more detailed discussion of Guha et al. [2004] can be found in Section 6).

4. EXTENDED PROBABILISTIC WAVELET SYNOPSES FOR RELATIVE ERROR

Unfortunately, conventional and extended wavelet synopses optimized for overall L_2-error metrics (as described in Sections 2–3), may not always be the best choices for approximate-query processing systems. As observed in the recent

work of Garofalakis and Gibbons [2004], conventional L_2-optimized 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 nontrivial guarantees for individual approximate answers. To address these shortcomings, their work introduces *probabilistic wavelet synopses*, a novel approach for constructing (single-measure) wavelet data summaries optimized for maximum relative-error metrics in the approximate data reconstruction. Given the pitfalls and shortcomings of synopses optimized for overall L_2-errors, it is obviously important to extend the ideas of (relative-error optimized) probabilistic wavelet synopses to the setting of *multimeasure* data and extended wavelet coefficients. This turns out to be a challenging problem, mandating novel algorithmic solutions (we also discuss the application of our ideas to deterministic relative-error thresholding in Section 4.5). Before delving into the details of our approach, we provide some necessary background material on (single-measure) probabilistic wavelets [Garofalakis and Gibbons 2004] (due to space constraints, our development here focuses primarily on the one-dimensional case; extensions to multidimensional wavelets are described in the Electronic Appendix).

4.1 Probabilistic Wavelet Synopses for Single-Measure Data

Consider the wavelet-transform array W_A containing wavelet coefficients for an input data vector. Rather than deterministically retaining the largest coefficients (in absolute normalized value) in a data summary, a probabilistic wavelet synopsis is constructed using a *probabilistic thresholding* process. In a nutshell, the key idea is to assign each nonzero coefficient c_i a *fractional amount of storage* $y_i \in (0, 1]$, essentially representing the probability of retaining the coefficient in the synopsis. A probabilistic wavelet synopsis is then built by flipping independent, appropriately biased coins for each nonzero c_i to decide whether the coefficient will be represented in the synopsis (or assumed to be zero). Given a set of fractional storage assignments $\{y_i\}$, it is easy to see that the expected size of the synopsis (over all possible coin-flip sequences) is simply $\mathrm{E}[|\,\text{synopsis}\,|] = \sum_{i|c_i\neq 0} y_i = \sum_{i|c_i\neq 0} \frac{c_i}{\lambda_i}$.

Garofalakis and Gibbons [2004] propose several different algorithms for building probabilistic wavelet synopses. The key, of course, is to select the coefficient fractional-storage assignment $\{y_i\}$ and the values to be retained in the synopsis 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., $\mathrm{E}[|\text{synopsis}|] \leq B$). They propose two winning strategies based on dynamic programming (DP) recurrences over the Haar error-tree structure for minimizing appropriate *probabilistic error metrics*. The rationale is that these metrics are directly related to the maximum relative-error (with an appropriate *sanity bound* s)[5] in the approximation of individual data values based on the synopsis. In other words, their schemes try to (probabilistically) control

[5]The role of the sanity bound is to ensure that relative-error numbers are not unduly dominated by small data values [Garofalakis and Gibbons 2004; Vitter and Wang 1999].

the quantity $\max_i\{\frac{|\hat{d}_i - d_i|}{\max\{d_i, \mathrm{S}\}}\}$, where $\hat{d}_i$ denotes the data value reconstructed based on the wavelet synopsis. Note, of course, that $\hat{d}_i$ is a *random variable*, defined as the ± 1 summation of all (independent) random coefficient selections on $\mathtt{path}(d_i)$. Bounding the maximum relative-error in the approximation also allows meaningful error guarantees to be provided on reconstructed data values [Garofalakis and Gibbons 2004].

The first algorithm of Garofalakis and Gibbons [2004], termed MinRelVar, insists on *unbiased* reconstruction for individual data values (i.e., $\mathrm{E}[\hat{d}_i] = d_i$) by requiring an unbiased approximation for each coefficient (again, in our discussion here, expectation is always defined over the space of possible coin-flip sequences). This is achieved by independently rounding each nonzero coefficient c_i either up to $\frac{c_i}{y_i}$ (with probability y_i) or down to zero. This implies that the variance for each coefficient $c_i \neq 0$ in the probabilistic synopsis is $\mathrm{Var}(i, y_i) = \frac{1-y_i}{y_i}c_i^2$. The MinRelVar dynamic program then tries to minimize the *maximum normalized standard error* (NSE) across all reconstructed values in the data domain. Their second algorithm, termed MinRelBias, relaxes the unbiasedness requirement, thus allowing the actual coefficient values c_i to be retained in the synopsis (with probability y_i). The expectation for each coefficient c_i in the synopsis hence becomes $c_i \cdot y_i$, which implies a bias of $c_i(1 - y_i)$ contributed to all data values under c_i's subtree. The MinRelBias DP algorithm attempts to minimize the *maximum normalized bias* in the data reconstruction.[6]

As an example, Eq. (1) depicts the key MinRelVar DP recurrence for minimizing the maximum NSE, defined as

$$\max_i \mathrm{NSE}(\hat{d}_i) = \max_i \frac{\sqrt{\mathrm{Var}(\hat{d}_i)}}{\max\{|d_i|, \mathrm{S}\}},$$

where $\mathrm{Var}(\hat{d}_i) = \sum_{c_j \in \mathtt{path}(d_i)} \mathrm{Var}(j, y_j)$. Here, $R[i, B]$ denotes the minimum 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 that a space budget of B and $\mathrm{Norm}(i) = \max\{d_{\min_i}^2, \mathrm{S}^2\}$ (where $d_{\min_i}$ is the minimum data value under c_i's subtree) is a normalization term for this 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 Eq. (1) states that for a given space budget B at c_i, the optimal fractional-storage allotments $\{y_k\}$ and 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 this 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 $B - y_i$ space "units" amongst the two child subtrees of c_i. Of course, if $c_i = 0$, then no space budget needs

[6]As with all randomized estimators, error is due to two key factors, namely, bias and variance. While unbiasedness is typically a desirable estimator feature, slightly biased estimators with smaller variance may actually perform better in practice [Cochran 1977].

to be allocated to node i, which results in the simpler recurrence in the second clause of Eq. (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 nonzero coefficient, but zero budget ($c_i \neq 0$ and $B = 0$).

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

The DP recurrence in Eq. (1) characterizes the optimal solution to the maximum NSE minimization problem for the case of *continuous* fractional-storage allotments $y_i \in (0, 1]$ (modulo certain technical conditions that may require small "perturbations" of zero coefficients) [Garofalakis and Gibbons 2004]. A similar DP recurrence can also be given for the maximum normalized bias metric. The authors' MinRelVar and MinRelBias algorithms then proceed by *quantizing the solution space*; that is, they assume that the storage allotment variables y_i and b_L in Eq. (1) take values from a discrete set of choices corresponding to integer multiples of $1/\mathrm{q}$, where $\mathrm{q} > 1$ is an input integer parameter to the algorithms (larger values of q imply results closer to the optimal, continuous solution). The running time of their (quantized) MinRelVar and MinRelBias algorithms is $O(N\mathrm{q}^2 B \log(\mathrm{q}B))$ with an overall space requirement of $O(N\mathrm{q}B)$ (and an in-memory working-set size of $O(\mathrm{q}B \log N)$); furthermore, their techniques also naturally extend to multidimensional data and wavelets, with a reasonable increase in time and space complexity [Garofalakis and Gibbons 2004]. Experimental results in Garofalakis and Gibbons [2004] have demonstrated the superiority of MinRelVar and MinRelBias probabilistic synopses as an approximate query answering tool over conventional wavelet synopses.

4.2 Extended Probabilistic Wavelets for Multiple Measures: Problem Formulation

In what follows, we introduce algorithms for building effective probabilistic synopses comprising extended wavelet coefficients for multimeasure datasets. As in Garofalakis and Gibbons [2004], our primary focus is on synopses that minimize the maximum relative-error (with appropriate sanity bounds) in the data reconstruction.[7] Employing the more complex extended-coefficient format enables effective space utilization, but at the same time, significantly increases the complexity of the probabilistic thresholding process, rendering the DP schemes of Garofalakis and Gibbons [2004] inapplicable in our problem setting.

The problem with extended coefficients. In a nutshell, the key difficulty in probabilistic thresholding for extended wavelet coefficients stems from the common header space (i.e., coordinates + bitmap) for all stored coefficient values.

[7]Our techniques also naturally extend to other approximation-error metrics, including maximum weighted relative error and maximum absolute error.

The ability to share this header is the main benefit of the extended-coefficient storage format, but at the same time, this sharing of storage introduces nontrivial dependencies in the thresholding process across coefficients for different measures, and implies that the selection probabilities for such coefficients are no longer independent. More formally, consider a dataset 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 (i.e., fractional storage) 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 as equal to the space required to store a single coefficient value (e.g., `sizeof(float)`), and all space requirements are expressed in terms of this unit). The expected space requirement of extended coefficient EC_i can be computed as

$$\mathrm{E}[|EC_i|] \;=\; \mathrm{sp}(y_{i1}, \ldots, y_{iM}) \;=\; \sum_{j\,|c_{ij}\neq 0} y_{ij} + H \times \left(1 - \prod_{j=1}^{M} (1 - y_{ij})\right). \qquad (2)$$

The first summand in the previous formula captures the expected space for all (nonzero) 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 are stored.

Eq. (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 storage dependencies across groups of coefficient values (corresponding to different measures). This requirement significantly complicates the design of probabilistic thresholding schemes 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 $|\hat{d}_{ij} - d_{ij}| \leq \epsilon \cdot \max\{|d_{ij}|, \mathrm{s}_j\}$ for given per-measure sanity bounds $\mathrm{s}_j > 0$, 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 [Garofalakis and Gibbons 2004], it is easy to see that *minimizing the overall* NSE *across all measures* guarantees a maximum relative-error bound which 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 fractional-storage assignments y_{ij} for coefficients c_{ij} that minimize the maximum NSE for each reconstructed data value across all measures; that is,

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

subject to the constraints $0 < y_{ij} \leq 1$ for all nonzero 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 Eq. (2).

We focus on the preceding maximum NSE minimization problem for multimeasure data in the remainder of this section; we do note, however, that our techniques and algorithms also naturally extend to multimeasure variants of the maximum normalized-bias minimization problems of Garofalakis and Gibbons [2004]. Instead of calculating in all the presented formulas/recurrences the variance of a coefficient c_i based on its retention probability y_i (as in the maximum NSE minimization problem), this extension requires computing the corresponding maximum normalized bias introduced by coefficient c_i, which is estimated as $\frac{|c_i|\times(1-y_i)}{\max\{|d_i|,\mathrm{s}\}}$. This is the only required modification, along with retaining the actual value of a coefficient c_i in the synopsis (instead of the "rounded-up" $\frac{c_i}{y_i}$) [Garofalakis and Gibbons 2004].

Our algorithms exploit both the error-tree structure of the Haar decomposition and the aforementioned storage dependencies (Eq. (2)) for extended coefficients in order to intelligently assign fractional storage $\{y_{ij}\}$ to nonzero coefficients within the overall space-budget constraint B. As in Garofalakis and Gibbons [2004], our schemes also rely on quantizing the space allotments to integer multiples of $1/\mathrm{q}$, where $\mathrm{q} > 1$ is an integer input parameter; that is, we modify the constraint $0 < y_{ij} \leq 1$ to $y_{ij} \in \{\frac{1}{\mathrm{q}}, \frac{2}{\mathrm{q}}, \ldots, 1\}$ in the earlier problem formulation (remember that our space unit corresponds to the size of a coefficient value). Our first algorithm is based on an *exact*, generalized DP formulation that extends earlier schemes for the single-measure case [Garofalakis and Gibbons 2004] to the multimeasure setting; unfortunately, this generalization comes at the cost of a significant increase in computational complexity (as our empirical study also clearly shows). Our second algorithm is a very fast, greedy approximation heuristic (termed GreedyRel) for probabilistic multimeasure thresholding; our results show that GreedyRel consistently provides near-optimal performance and can easily scale to problem sizes that are simply unattainable for DP-based solutions (even for the simpler single-measure case!).

4.3 PODPRel: An Optimal Partial-Order Dynamic Programming Solution

Consider an input dataset with M measures. At a high level, our maximum NSE minimization problem for extended wavelet coefficients (Section 4.2) can be seen as a generalization of the single-measure NSE minimization setting of Garofalakis and Gibbons [2004], where our final goal is to minimize *the maximum*

component of an M-component vector of NSEs (i.e., those NSE values for the approximation of each individual measure). Of course, the key complication here 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 intricate storage dependencies arising between the approximation at different measures because of shared header space (Eq. (2)). As already discussed in Section 4.2, 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.

Consider once again the DP recurrence in Eq. (1) for the single-measure case. Remember that the recurrence computes in $R[i, B]$ the minimum value of the NSE^2 among all data values under coefficient i in the error tree, assuming a space budget of B. Based on our previous discussion, extending the formulation to the case of multiple measures requires that we generalize $R[i, B]$ to denote an M-component vector of NSE^2 values corresponding to all M measures for data values in the subtree rooted at the coefficient with coordinate i, and assuming a total space of B allotted to extended coefficients in this subtree. We similarly generalize Var and Norm in Eq. (1) to the M-component vectors of per-measure variances $\text{Var}(i, y_{ij})$ and normalization values $\text{Norm}(i, j)$ at extended coefficient i (for a total allotment of y_i); also, addition and division operators denote corresponding component-wise operations on the operand vectors. As in the single-measure case, we can simplify the minimization problem of Eq. (3) by normalizing the variance value at each node with the normalization terms $\text{Norm}(i, j) = \max\{d^2_{\min_{ij}}, \mathtt{s}^2_j\}$ of its subtrees. Thus, we compute the jth component of the $R[i, B]$ vector 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 as

$$R[i, B]_j = \max \left\{ \begin{array}{l} \frac{\text{Var}(i, y_{ij})}{\text{Norm}(2i, j)} + R[2i, b_{2i}]_j \\ \frac{\text{Var}(i, y_{ij})}{\text{Norm}(2i+1, j)} + R[2i+1, b_{2i+1}]_j \end{array} \right. .$$

The computation of $R[i, B]$ at node i iterates over all fractional storage M-tuples $(y_{i1}, \ldots, y_{iM}) \in \{0, \frac{1}{\mathtt{q}}, \frac{2}{\mathtt{q}}, \ldots, 1\}^M$ and subtree space allotments such that $b_{2i} + b_{2i+1} + \text{sp}(y_{i1}, \ldots, y_{iM}) = B$, where $\text{sp}(y_{i1}, \ldots, y_{iM})$ denotes the expected space requirements of the extended coefficient at node i for the given fractional storage M-tuple (Eq. (2)). Our goal, of course, is to minimize the maximum component of the vector $R[\mathtt{root}, B]$; that is, minimize $\max_{k=1,\ldots,M}\{R[\mathtt{root}, B]_k\}$.

Unfortunately, this generalization of $R[i, B]$ to an M-component vector also implies that (so as to ensure optimality) the bottom-up computation of the DP recurrence can no longer afford to maintain just the locally-optimal partial solution for each subtree (as in the single-measure case). 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 no longer sufficient; more information needs to be maintained and explored during the bottom-up computation.

Consider a simple scenario with $M = 2$, and (slightly abusing notation) let $R[2i, B-y] = [2.5, 2]$ and $R'[2i, B-y] = [1, 3]$ 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 $R'[]$). Finally, let the normalized variance vector for the coefficient values at node i, assuming a total space of y, be $\frac{\text{Var}(i,y)}{\text{Norm}(2i)} = \frac{\text{Var}(i,y)}{\text{Norm}(2i+1)} = [2, 0.5]$. It is easy to see that in this case, even though $R'[2i, B-y]$ is locally suboptimal at node $2i$ (since its maximum component is larger than that 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.

The key here is that unlike conventional dynamic programming, using an M-component vector $R[i, B]$ to capture the per-measure squared NSEs corresponding to different partial solutions in the DP computation also means that the conventional principle of optimality based on a total ordering of partial solutions [Cormen et al. 1990] is no longer applicable. Thus, locally suboptimal $R[i, B]$'s (i.e., with large maximum component NSE^{2}s) cannot be safely pruned, since they may in fact be part of an optimal 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:

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

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.

Our proposed *partial-order dynamic programming (PODP)* solution to the maximum NSE minimization problem for extended coefficients (termed PODPRel) generalizes the corresponding DP formulation in Garofalakis and Gibbons [2004] based on the preceding observations.[8] In other words, 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 PODPRel 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, PODPRel needs to iterate over all partial solutions computed in $\mathcal{R}[i, B]$ so as 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

[8]Ganguly et al. [1992] also discuss PODP in a completely different context, namely, designing a System-R-style algorithm for optimizing join orders in parallel database systems.

all possible space allotments $\{y_{ij}\}$ to each individual measure and estimate the overall space requirements of the extended coefficient using Eq. (2).

The main drawback of our PODP-based solution is a dramatic increase in time and space complexity compared to the single-measure case. PODPRel relies on a much more strict, partial-order criterion for pruning suboptimal solutions, which implies that the sets of incomparable partial solutions $\mathcal{R}[i, B]$ which 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 $O(\mathrm{q}^M)$, compared to only $O(\mathrm{q})$ 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.

4.4 GreedyRel: An Efficient, Greedy Approximation Heuristic

Given the very high running-time and space complexities of our PODP-based solution, 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 that *subset of* extended wavelet coefficients in the error tree which results 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 that subset of coefficients in the subtree which 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. Consider a step of our GreedyRel algorithm, and let y_{ij} denote the currently assigned retention probability (i.e., fractional storage) for each individual coefficient c_{ij} (i.e., at coordinate i for the j^{th} measure); 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

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

The preceding formula is similar to the DP recurrence for computing $R[i, B]$ in Eq. (1): 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. Note, however, that while Eq. (1) is exact for the maximum squared NSE in the optimal, continuous solution for single-measure data [Garofalakis and Gibbons 2004], the aforementioned 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 in its computation.

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 those coefficient values in T_{ij} which, when allotted additional space quanta, 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 synopsis space-budget in space quanta of $1/\mathrm{q}$, where $\mathrm{q} > 1$). Let $G[i, j]$ be the current estimated maximum NSE^2 for T_{ij} (as described before), and let $G_{\mathsf{pot}}[i, j]$ denote the *potential* estimated maximum NSE^2 for T_{ij}, assuming that a (minimal) additional space of $1/\mathrm{q}$ is allotted to all coefficient values in potSet$[i, j]$. Also, let potSpace$[i, j]$ denote the increase in the overall synopsis size, that is, 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 Eq. (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 potSpace$[i, j]$ for all coefficient values in potSet$[i, j]$ can be obtained by adding the results of Eq. (4) for each of these values (with $\delta y_{kj} = \frac{1}{\mathrm{q}}$); that is,

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

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

To estimate the potSet$[i, j]$ sets and the corresponding $G_{\mathsf{pot}}[i, j])$ (and 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 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 $G_{\mathsf{pot}}[i, j] = \frac{\mathrm{Var}(c_{ij}, y_{ij}+\frac{1}{q})}{\mathrm{Norm}(i,j)}$.[9] For a nonleaf coefficient c_{ij}, GreedyRel considers three distinct cases of forming potSet$[i, j]$ and selects the one resulting in the largest marginal error-gain estimate: (1) potSet$[i, j] = \{c_{ij}\}$ (i.e., select only c_{ij} for additional storage); (2) potSet$[i, j]$ = potSet$[k, j]$, where $k \in \{2i, 2i + 1\}$ is such that $G[i, j] = G[k, j]+$ Var(c_{ij}, y_{ij})/Norm(k, j) (i.e., select that potSet from the child subtree whose estimated maximum NSE^2 determines the current maximum NSE^2 estimate at c_{ij}); and (3) potSet$[i, j]$ = potSet$[2i, j]\cup$ potSet$[2i + 1, j]$ (i.e., select the union of the potSets from both child subtrees). Among the aforementioned three choices, GreedyRel selects the one resulting in the largest value for gain(potSet$[i, j]$), and records the choice made for coefficient c_{ij} (1, 2, or 3) in a variable choice$_{ij}$.[10] In order to estimate gain(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), described earlier, and $l = \{2i, 2i + 1\} - \{k\}$):

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

As an example, consider the scenario depicted in Figure 5 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. 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 that choice which 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 value. 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, regardless of how much it would reduce the maximum NSE^2 value through its

[9]As in Garofalakis and Gibbons [2004], 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 nonzero allotments $y_{ij} > 1/2$ to coefficient c_{ij}.

[10]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.

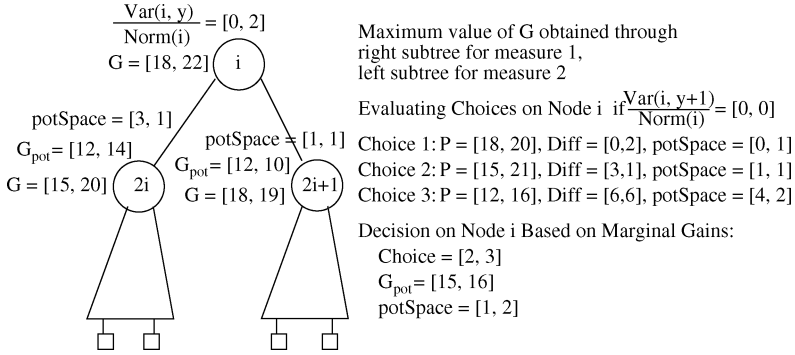

Fig. 5. Example scenario for the GreedyRel algorithm.

subtree. This happens because the estimated value of $G[i, j]$ through the other subtree would remain the same. Note that in single-measure datasets, the value of G through both subtrees is the same in the optimal solution [Garofalakis and Gibbons 2004], implying that the preceding 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 choice$_{ij}$ variable at each node c_{ij}, and continuing along the appropriate subtrees of the node until we reach nodes with choice$_{ij} = 1$.

Distributing the available synopsis space. After completing the aforementioned steps, our GreedyRel algorithm has computed the estimated current and potential maximum NSE^2 values $G[0, j]$ and $G_{\text{pot}}[0, j]$ (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 choice$_{ij_{\max}} = 1$, set $y_{ij_{\max}} := y_{ij_{\max}} + \frac{1}{q}$; (2) if choice$_{ij_{\max}} = 2$, 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 choice$_{ij_{\max}} = 3$, then recurse to both child subtrees T_{2i} and T_{2i+1}; furthermore, after each of the previous steps, compute the new values for $G[i, j_{\max}]$, $G_{\text{pot}}[i, j_{\max}]$, potSpace$[i, j_{\max}]$, and choice$_{ij_{\max}}$ at node i.

A pseudocode description of our GreedyRel algorithm is depicted in Figure 6. Note that in 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 choice$_{ij_{\max}} = 2$), GreedyRel first recurses on that child causing the maximum estimated squared NSE, and then recurses on the other child with any remaining space (Lines 12–16 of traverse).

procedure GreedyRel(W_A, B, q, S)
Input: $N \times M$ array W_A of Haar wavelet coefficients; space constraint B; quantization parameter $\mathsf{q} > 1$; 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 $G[i,j]$, $G_{\mathsf{pot}}[i,j]$, potSpace$[i,j]$, and choice$_{ij}$
5. spaceLeft $= B$
6. **while** (spaceLeft > 0) **do**
7. $j_{\max} := \arg\max_j \{G[0,j]\}$
8. occupiedSpace := traverse(0, $j_{\max}$, q, y, spaceLeft)
9. spaceLeft := spaceLeft - occupiedSpace
10. **if** (occupiedSpace = 0) **then return**(y) *//not enough space*
11. **endwhile**
12. **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 $\mathsf{q} > 1$; array y of current retention probabilities; maximum synopsis space to allocate.
Output: Space allocated to the T_{ij} subtree at this step.
begin
1. allocatedSpace := 0
2. **if** (choice$_{ij} = 1$) **then**
3. neededSpace := $\delta_j(\mathrm{E}[|EC_i|], 1/\mathsf{q})$ *// see Eq. (4)*
4. **if** (neededSpace $\leq$ spaceLeft) **then**
5. $y_{ij} := y_{ij} + 1/\mathsf{q}$
6. allocatedSpace := neededSpace
7. **endif**
8. **else if** (choice$_{ij} = 2$) **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 $G[i,j]$, $G_{\mathsf{pot}}[i,j]$, potSpace$[i,j]$, and choice$_{ij}$
19. **return**(allocatedSpace)
end

Fig. 6. The GreedyRel algorithm.

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 choice$_{ij}$. Thus, the space requirements per node are $O(M)$, resulting in a total space complexity of $O(NM)$.

In the bottom-up initialization phase (Lines 1–4), GreedyRel computes, for each error-tree node, the values of the $G[i, j]$, $G_{\mathsf{pot}}[i, j]$, potSpace$[i, j]$, and choice$_{ij}$ 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 $\geq K \times 1/\mathrm{q}$ (see Eq. (4)). To reach these K coefficients, GreedyRel traverses exactly K paths of maximum length $O(\log N)$. For each visited node and just for the selected measure $j_{\max}$ (chosen at the root), we need to compute the new values of G, G_{pot}, potSpace, and choice, which requires $O(1)$ time. Finding the measure $j_{\max}$ with the maximum estimated NSE^2 value at the root requires time $O(\log M)$.[11] Thus, overall, GreedyRel distributes space $\geq K \times 1/\mathrm{q}$ in time $O(K \log N + \log M)$, making the amortized time per-space quantum $1/\mathrm{q}$ equal to $O(\log N + \log M/K) = O(\log(NM))$. Since the total number of such quanta that we need to distribute is $B\mathrm{q}$, the overall running time complexity of GreedyRel is $O(NM + B\mathrm{q}\log(NM))$.

4.5 Discussion: Deterministic Relative-Error Thresholding

In order to avoid the potential pitfalls of probabilistic thresholding schemes (e.g., "bad" coin-flip sequences), recently, Garofalakis and Kumar [2005] have proposed optimal *deterministic* thresholding algorithms for relative-error metrics. Their key observation is that while such error measures do not have the simple monotonic/additive structure of probabilistic NSE^2 over the coefficient error-tree, an optimal deterministic synopsis can be constructed using a novel DP formulation which also considers choices made at ancestors of the current node. More specifically, the basic idea is to condition the optimal error value for a subtree not only on the root node of the subtree c_i and the amount B of synopsis storage allotted (Eq. (1)), but also on the *subset of coefficients* S selected on the path from the error-tree root to node c_i (i.e., considering all possible $S \subseteq \mathtt{path}(c_i)$). Since the depth of the error tree is $O(\log N)$, it is possible to tabulate all such possible selections while retaining an $O(N^2)$ running time for the DP algorithm (for one-dimensional data) [Garofalakis and Kumar 2005].

It is not difficult to see that similar ideas can also be applied to our PODP formulation (Section 4.3) to give a deterministic maximum relative-error minimization scheme for M-measure data. Since our objective is still the maximum over an M-component vector of per-measure maximum relative errors, as in Section 4.3, our deterministic algorithm cannot prune locally-suboptimal partial solutions and needs to tabulate a (potentially large) collection of incomparable solutions based on the M-component less-than partial order $\preceq_M$ at each node. Thus, the fundamental complexity issues of the PODP formulation remain in the deterministic scheme. In addition, conditioning on the coefficient selections on the path to the current error-tree node in order to produce a deterministic PODP recurrence only exacerbates the problem: Node i now must tabulate a collection of incomparable solutions $\mathcal{R}[i, B, S]$ *for each possible selection* S *of extended coefficients* on $\mathtt{path}(c_i)$. Since there are a total of 2^M possible choices at each ancestor node (any subset of up to M nonzero coefficients), this implies that the space/time complexities of the deterministic PODP formulation increase by a multiplicative factor of $2^{M \cdot O(\log N)} = O(N^M)$, essentially rendering such a deterministic solution unusable for any realistic problem sizes.

[11] Just for the root node, we may store the $G[0, j]$ values in a heap.

The complexity of conditioning on ancestor-coefficient choices also explodes when extending the optimal DP formulation to multidimensional data (even for the single-measure case [Garofalakis and Kumar 2005]). Ideas from the approximate "sparse" DP schemes proposed in Garofalakis and Kumar [2005] can potentially be employed to limit the search space of our multimeasure deterministic PODP scheme at the cost of an approximation-error bound; still, the issue of maintaining and exploring multiple incomparable solutions at each node remains. Designing fast, heuristic algorithms, for instance, by extending the recently proposed greedy heuristics of Karras and Mamoulis [2005] to the multimeasure case is another possibility. In general, the problem of designing practical, multimeasure deterministic thresholding schemes for relative-error metrics is complex and left open as an interesting area for future work.

5. EXPERIMENTAL STUDY

In this section, we present an extensive experimental study of our proposed algorithms for constructing wavelet synopses over datasets with multiple measures. Besides validating the effectiveness of our extended wavelet coefficient approach against the existing Individual and Combined schemes, one of the main objectives of our study was to evaluate the accuracy and scalability of our proposed GreedyL2 and GreedyRel algorithms over several real-life and synthetic multimeasure datasets. The main findings of our study can be summarized as follows.

—*Highly scalable solutions*: Our GreedyL2 and GreedyRel algorithms provide fast and highly scalable solutions for constructing conventional and probabilistic wavelet synopses over large multimeasure datasets. Unlike earlier schemes (and the DynProgL2 and PODPRel algorithms), the GreedyRel and GreedyL2 algorithms exhibit a linear dependency on the domain size. Moreover, for probabilistic wavelet synopses, the running time and space requirements of earlier techniques yield our GreedyRel algorithm as the only viable solution, even for the single-measure case, for large real-life datasets.

—*Near-optimal results*: The GreedyL2 and GreedyRel algorithms consistently provide near-optimal solutions when compared to DynProgL2 and PODPRel (respectively), demonstrating that they constitute efficient techniques for constructing accurate conventional and probabilistic synopses over large multimeasure datasets.

—*Improved accuracy for individual reconstructed answers through the use of extended wavelet coefficients*: Compared to earlier approaches that operate on each measure individually, our GreedyL2 and GreedyRel algorithms significantly reduce the weighted sum-squared and maximum relative-error of the approximation. These improvements are often by a factor of two–three, but in many cases, we also observe up to seven times smaller errors than the closest competitive technique. The improvements in the obtained accuracy are, of course, due to the flexible storage format of the extended wavelet coefficients and the improved storage utilization that they achieve.

All experiments reported in this section were performed on a personal computer using an Athlon XP 1800+ processor with 512MB of RAM memory.

5.1 Techniques and Parameter Settings

Our experimental study is split into two parts, based on the error metric that our algorithms try to minimize. We here need to emphasize that besides the presented techniques, we also performed a comparative analysis of the GreedyL2 and GreedyRel algorithms in multimeasure datasets. However, the results were, as expected, qualitatively similar to those presented in Garofalakis and Gibbons [2004], with the GreedyL2 algorithm producing wavelet synopses with smaller weighted sum-squared errors, while the GreedyRel algorithm consistently produced significantly smaller maximum relative-errors. This is not surprising, as the two algorithms are designed to minimize different error metrics, and the existence of multiple measures in datasets cannot result in a qualitatively different behavior than in single-measure datasets. We thus omit this comparison from our discussion.

—*Weighted sum-squared error.* We initially compared the performance of the GreedyL2 and DynProgL2 algorithms for constructing conventional wavelet synopses over multimeasure datasets to the following four algorithms: (1) random sampling (RS) using the Reservoir algorithm described in Vitter [1985], since the datasets that we used did not contain duplicate tuples; (2) *Ind*, where the individual space allocated to each measure is proportional to its weight, and the Individual algorithm is then run for each measure; (3) *IndSorted*, where the individual coefficients from all measures are sorted according to their weighted benefit, and those with the highest benefits are retained, without imposing any limit on the size allocated to each measure; and (4) Combined, where the *combined* coefficients are sorted according to their overall weighted benefit, and those with the highest benefits retained. As input to our algorithms, we used the output of the decomposition step of the Combined algorithm, which we found to produce better results than the corresponding output of the Individual algorithm.

—*Maximum relative-error.* In the second part of our experimental study, we compare our GreedyRel and PODPRel algorithms for constructing probabilistic data synopses over multimeasure datasets, along with a technique, which we will term IndDP, that partitions the available space equally over the measures and then operates on each measure individually by utilizing the dynamic programming MinRelVar algorithm [Garofalakis and Gibbons 2004]. To provide a more fair comparison to the IndDP algorithm, the majority of our experiments include datasets where all the measured quantities exhibit similar characteristics, thus yielding uniform partitioning of the synopsis space over all measures as the appropriate space allocation technique. The only parameter in our algorithms is the quantization parameter q, which is assigned a value of ten for the GreedyRel and IndDP algorithms, and a smaller value of four for the PODPRel algorithm to reduce its running time (the accuracy of the produced synopses was similar in PODPRel with larger values of q). Moreover, the sanity bound of each measure is set to the 10%-quantile value of the measure's data values.

Table V. Data Generator Input Parameters and Default Values

Parameter	Description	Default Value
D	Number of dimensions	2
M	Number of measures	6
$Card_i$	Cardinality of dimension i	1,024
$n_{regions}$	Number of dense regions	10
V_{min}, V_{max}	Minimum and maximum volume of regions	4,900,4,900
Z	Skew across regions	0.5
z_{min_i}, z_{max_i}	Minimum and maximum skew within region i	1, 1
Sum_i	Sum of values for measure i	1,000,000
$spCount$	Fraction of populated cells in sparse areas	0.05
$spSum_i$	Sum of values of populated cells in sparse area i	0.05

Table VI. Data Generator Value Distributions

Distribution	Description
NoPerm	Cells with smaller L1-distance from lower-left corner have larger values
Normal	Cells with smaller L1-distance from center have larger values
PipeOrgan	Cells with smaller L1-distance from center have smaller values
Middle	Consider a hyper-rectangle centered at the region's center, and having, for each dimension, half the length of the corresponding region length. Cells with smaller L1-distance from this hyper-rectangle have larger values
Altered-X	This measure follows the same distribution as X distribution, but its values are randomly altered by up to 50%

5.2 Datasets

We experiment with several one-dimensional and multidimensional synthetic multimeasure datasets and present in this section a representative set of results. The input parameters of our Zipfian data generator, along with their default values, are described in Table V. The generator begins by populating $n_{regions}$ rectangular regions of a D-dimensional array whose size is determined by the number of the dataset's dimensions and the cardinalities $Card_i$ of each dimension. The number of cells within each region is bound by the values V_{min} and V_{max}. The total sum Sum_i of values for each measure is partitioned across the $n_{regions}$ rectangular regions through the use of a Zipf function with parameter Z. Then, within each region, each measure's values are distributed by using one of the five distributions described in Table VI, with the parameter's values ranging from z_{min_i} to z_{max_i}. Notice the use of the *Altered-X* distribution ($X \in$ {NoPerm, Normal, Middle, PipeOrgan}), which helps create pairs of measures with similar, but not identical, data distributions. The data generator then also populates a number of cells in the remaining D-dimensional space, outside the dense regions. The fraction of such cells over the total number of populated cells is defined by the *spCount* parameter, and the total sum of values of these cells by the $spSum_i$ parameter. Especially for the case of one-dimensional datasets, the produced Zipfian distributions span the entire domain (i.e., $n_{regions} = 1$, $V_{min} = V_{max} = Card_1$ and $spCount = spSum_i = 0$).

In our experimental study, we also use a real-life dataset. Our `Weather` dataset contains meteorological measurements obtained by a station at the University of Washington (data at `http://www-k12.atmos.washington.edu/`

Table VII. Deviation Factor of GreedyL2 Benefit when Compared to the DynProgL2 Benefit

	Storage Constraint (Bytes)				
	1,200	2,400	3,600	4,800	6,000
Deviation Factor	3×10^{-5}	5×10^{-4}	10^{-6}	10^{-4}	10^{-6}

`k12/grayskies`). For this dataset, we extract the following six measured quantities: wind speed, wind peak, solar irradiance, relative humidity, air temperature, and dewpoint temperature.

Approximation-error metrics. The reported approximation-error metric in each case depends on our optimization problem. For conventional wavelet synopses, we mainly report the weighted absolute error for all queries of our workload, calculated as

$$\left(\sum_{j=1}^{M} W_j\right)^{-1} \times \sum_{i=1}^{M} (W_i \times |actualResult_i - estimatedResult_i|),$$

where the variables $actual\,Result_i$ and $estimated\,Result_i$ denote the exact and estimated values of the query result for measure i, correspondingly. The weighted sum-squared and relative errors are defined similarly, with the weighted sum-squared error also being reported in most cases. In the case of probabilistic wavelet synopses, we focus on the maximum relative-error of the approximation, which can provide guaranteed error-bounds for the reconstruction of any individual data value, and is the error metric that our probabilistic wavelet synopses algorithms try to minimize.

5.3 Weighted Sum-Squared Error Algorithms

5.3.1 *Synthetic Datasets.* We first investigate how close the weighted benefit achieved by the GreedyL2 algorithm is to that achieved by the DynProgL2 algorithm. We created a synthetic two-dimensional dataset with four measures, following Normal, Altered-Normal, Middle, and PipeOrgan distributions, and set the remaining parameters to the default values of Table V. We modified the storage constraint from 1,200 to 6,000 bytes and present the results in Table VII. The deviation factor presented in this table is defined as: $1 - \frac{Benefit(\text{GreedyL2})}{Benefit(\text{DynProgL2})}$. The GreedyL2 algorithm produced solutions with benefits very close to the optimal, as expected by its tight approximation bound. Due to the large amount of memory required by the DynProgL2 algorithm, we were unable to execute it for the remaining experiments and thus omitted this from the presented results.

We now evaluate the impact of several parameters on the performance of our GreedyL2 algorithm. In each experiment, unless specified otherwise, the data generator parameters were set to their default values. For the default number of measures (6), the data distributions were: Normal, Altered-Normal, PipeOrgan, Altered-PipeOrgan, Middle, and Altered-Middle. The query workload always consisted of 100 range queries, with the width of the range on each dimension being equal to ten. The queries targeted the dense areas with greater probability, since most of the data is stored there. The default storage bound was set to 5% of the dataset's size. Additional experiments can be found in the Electronic Appendix.

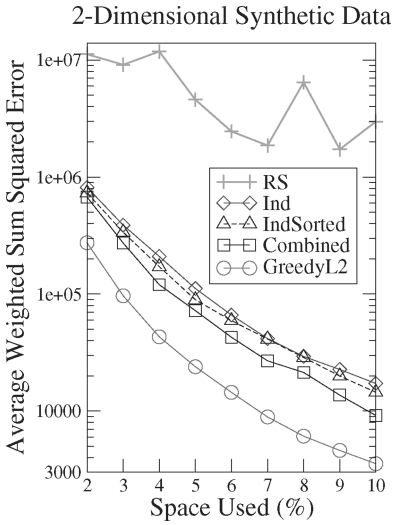

Fig. 7. Average weighted sum-squared error.

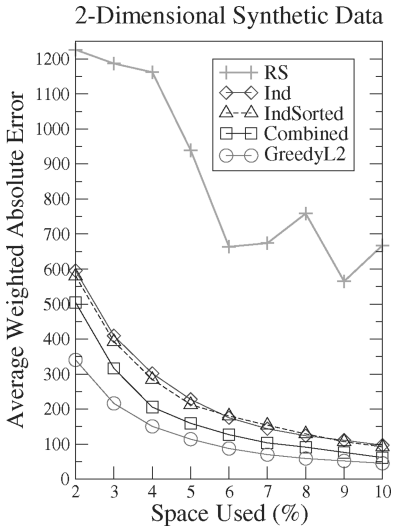

Fig. 8. Average weighted absolute error.

Storage space. In Figures 7 and 8, we present the average weighted sum-squared and absolute errors, respectively, for all algorithms as the storage space is varied from 2 to 10% of the dataset's size. The skew of the data distributions within each region was set to 1.5. Note that the y-axis of Figure 7 is logarithmic due to the large errors exhibited by random sampling. The GreedyL2 algorithm produced considerably smaller errors than the others. In particular, the average weighted sum-squared error of GreedyL2 was, in most cases, about 3 times smaller than the error of the closest competitor, and as low as 3.5 times smaller (6,105.25 versus 21,399.2 for 8% space). For the average weighted absolute

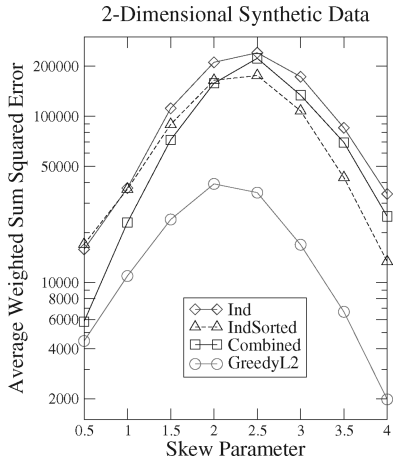

Fig. 9. Sensitivity to skew: sum-squared error.

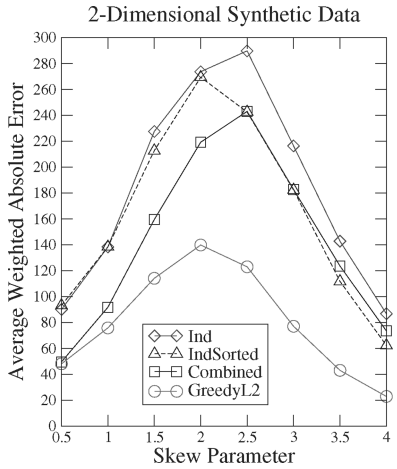

Fig. 10. Sensitivity to skew: absolute error.

error case, the error of GreedyL2 was typically about 1.5 times smaller than that produced by the closest competitor (51.85 versus 91.19 for 9% space, a ratio of 1.76). From the remaining methods, the Combined algorithm produced the best results. For the remaining experiments, we omit from the graphs the results for the random sampling algorithm, since its errors were consistently much larger than those of the other algorithms.

Skew within regions. We modified the Zipfian parameter controlling the skew of the measure's data distributions within each region from 0.5 to 4. Figures 9 and 10 present the obtained results for the weighted average sum-squared and absolute errors. As the skew increases, for each distribution, the coefficients with large values are limited to an increasingly smaller area. This results on one hand in the reduction of the sum-squared error of the results as the number of coefficient values that greatly influence it becomes smaller. On the other

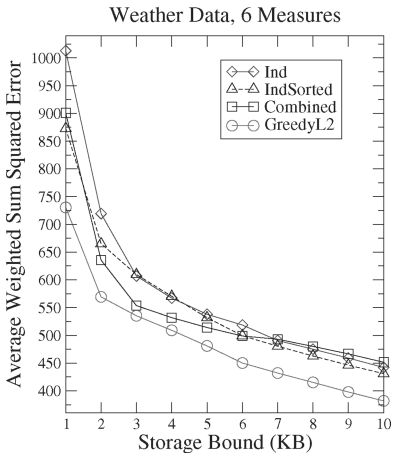

Fig. 11. Sum-squared errors for the weather dataset.

hand, the probability that coefficient values from multiple measures are simultaneously important is decreased. These two factors justify the relative improvement of the performance of IndSorted over the Combined algorithm for larger skew values. While the Combined algorithm performs closely to the GreedyL2 algorithm for small skews, the difference becomes very large as the skew increases. For large skews, GreedyL2 exhibits about a three-fold improvement over the closest competitive algorithm for the average weighted absolute error (22.79 versus 62.21 for skew parameter = 4) and up to a seven-fold improvement for the average weighted sum-squared error, for the same skew parameter.

5.3.2 *Real Dataset.* For our real dataset, we used the weather dataset containing weather measurements from the state of Washington (see Section 5.2). To simulate enhanced interest to specific measures, we assigned a weight value of 3 to the first measure, 2 to the next two measures, and 1 to the remaining measures. We constructed a two-dimensional dataset with the day and time of each measurement as the two dimensions, with a total of 521,817 tuples. We performed 1,000 range queries, where each range for the two dimensions was a random number with maximum values of 30 and 180, respectively. Thus, the maximum selectivity of a query was about 1%. From the results of each query, we calculated average values for stored measures over the queried day and time periods. The average values for each query were calculated by using the number of cells that each query accessed. All measures were normalized with the process described in the Electronic Appendix. Additional experiments involving different query selectivities and nonnormalized measures can be found in Deligiannakis and Roussopoulos [2003b]. In Figures 11 and 12, we present the results for the average weighted sum-squared and absolute errors, as the storage bound was varied from 1KB to 10KB. The errors for all wavelet methods decrease with the increase of the space bound. As can be seen from the two figures, the GreedyL2 algorithm consistently produced the smallest errors for

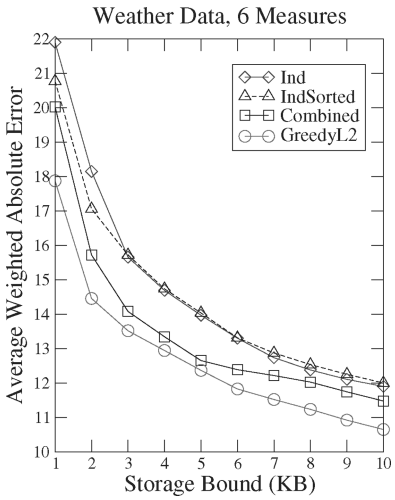

Fig. 12. Average weighted absolute errors for the weather dataset.

all error metrics. It is interesting to note that for the average weighted sum-squared error metric, the competitive techniques require at least 50% more space to achieve the same error as our GreedyL2 algorithm. However, the improvements of GreedyL2 in this case are smaller than in previous experiments where a similar behavior was observed (i.e., Figure 7), since in this case, the error decreases at a smaller rate. The average weighted sum-squared and absolute errors of random sampling were two and one orders of magnitude larger, respectively, than the corresponding errors of GreedyL2.

5.4 Maximum Relative-Error Algorithms

We now compare the performance and accuracy of our GreedyRel algorithm in comparison to the PODPRel and IndDP algorithms described in Section 5.1. Our study included several one-dimensional synthetic and real datasets. For the synthetic datasets, our Zipfian data generator produced 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. In all types of used Zipfian distributions, 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, one-third of the M distributions have the NoPerm shape, one-third 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.

For the derivation of the wavelet synopsis after selection of coefficient retention probabilities for either the GreedyRel or IndDP algorithm, we used the following method. We first performed ten trials using random coin-flips in order

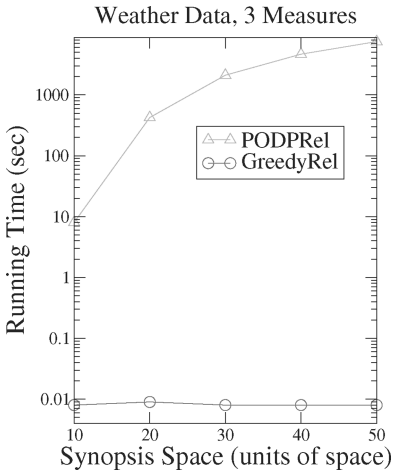

Fig. 13. Running time vs. space.

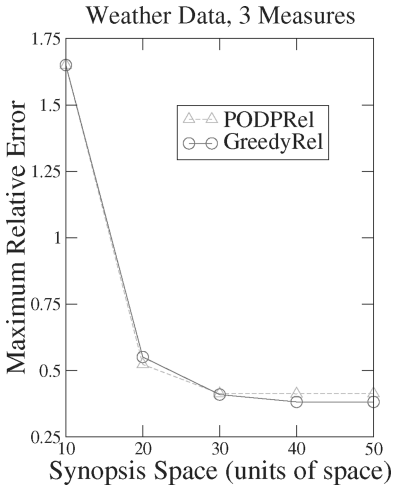

Fig. 14. Maximum relative-error.

to select which coefficient values to retain. For each trial, if the resulting synopsis size was below (above) the desired size constraint, then the coefficient values with the largest (smallest) retention probabilities which had not (respectively, had) already been selected for storage are included in (respectively, discarded from) the final synopsis. The selected synopsis is that which resulted in the smallest maximum relative-error (out of the ten trials).

Comparing PODPRel and GreedyRel. We now evaluate the accuracy and running time of the GreedyRel in comparison to the PODPRel algorithm. In Figures 13, 14, and 15 we plot the running time and maximum and average relative-errors, correspondingly, for the two algorithms and for the weather dataset 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

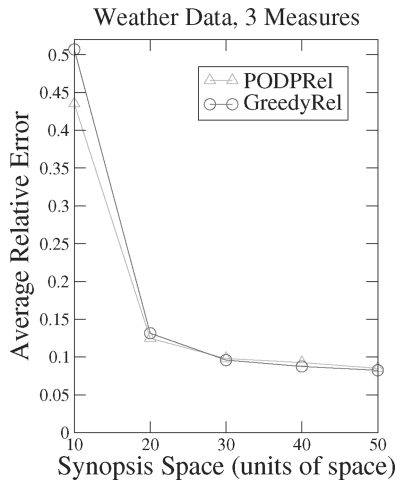

Fig. 15. Average relative-error.

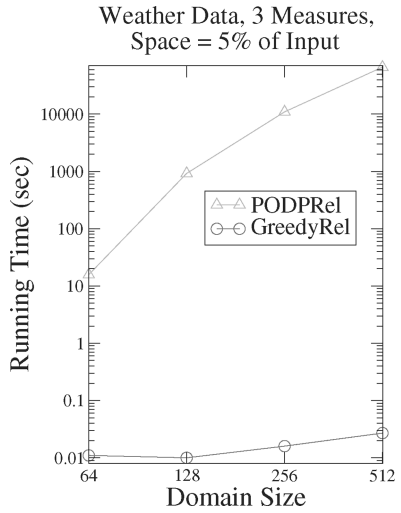

Fig. 16. Running time vs. domain.

experiment, we only use from the weather data the three measures most difficult to approximate. The domain size of the dataset 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 PODPRel algorithm does not scale well with the size of data synopsis, even for such a small dataset. For example, for a synopsis size of 50 space units, the PODPRel algorithm requires more than two hours to complete, while the GreedyRel algorithm required just a few milliseconds. However, as Figures 15 and 16 demonstrate, the GreedyRel algorithm provides near-optimal solutions in all cases.

In Figure 16 we present the corresponding running times for both algorithms as the domain size is increased from 64 to 512. From the weather dataset, we again extract just three measures, and set the synopsis space to always be 5%

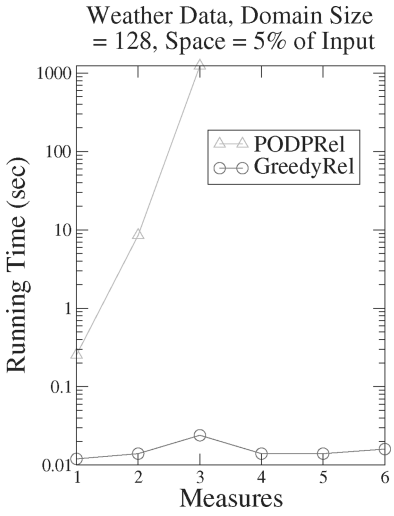

Fig. 17. Running time vs. measures.

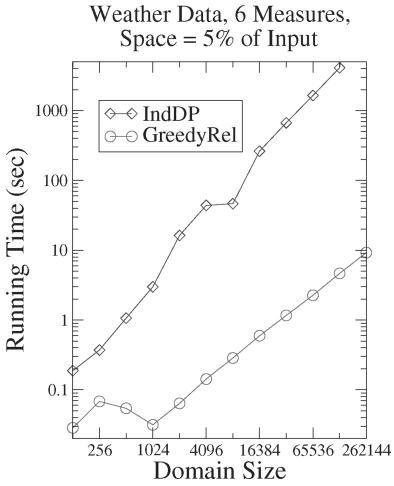

Fig. 18. Running time vs. domain size.

of the size of the input. Again, the running time performance of PODPRel is disappointing. For a domain size of 512, its running time exceeds 14 hours. Finally, as Figure 17 demonstrates, the running time of PODPRel increases exponentially with the number of dataset measures. Note that for datasets with four or more measures, PODPRel does not terminate within one day. It is easy to see that the PODPRel algorithm cannot be used but for toy-like datasets. 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 times. In Figure 18 we plot the running

times of the IndDP and GreedyRel algorithms for the weather dataset (all six measures were included) as the domain size is increased from 128 to 524,288. The synopsis size is always set to 5% of the input data. The IndDP is considerably slower than the GreedyRel algorithm (three orders of magnitude slower for a domain size of 131,072), with the difference increasing rapidly with the increase of domain size. Note that while the GreedyRel algorithm scales linearly with the increase in domain size (doubling the domain size doubles the running time), the IndDP 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 4.1), since when the domain size is doubled, the synopsis space is, as well. Moreover, the large memory requirements ($O(NB\text{q})$) 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 domain size, in terms of both its running time and memory requirements, constitutes it as the only viable technique (except for small datasets) for providing tight error guarantees, not only on multimeasure datasets, but also on single-measure datasets, since both the GreedyRel and IndDP algorithms scale in a similar way for such datasets. Moreover, as we will demonstrate in this section, the GreedyRel algorithm, which utilizes 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 datasets. For our synthetic datasets, we use a domain size of 16,384, 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 datasets. These six combinations arise from considering Zipfian distributions with skew 0.2 and 0.8, along with all other possible combinations of the used Zipfian distributions (AllNoPerm, AllNormal and Mixed). The synthetic datasets in this section contain six measures/distributions. The centers of these Zipfian distributions were randomly placed within an area of width equal to 128. The smaller (larger) the distance amongst distribution centers, the larger (smaller) the benefits that we expect from our techniques, since the possibility that multiple important coefficients will correspond to the same wavelet coordinates increases (decreases). In our experiments we examined two variations of our minimization algorithms. The unbiased version seeks to minimize the maximum normalized standard error for each reconstructed data value, while the biased version seeks to minimize the corresponding maximum normalized bias (see Section 4.1).

We first consider the possible combinations arising from distributions having skew equal to 0.2. In Figures 19 and 20 we plot the maximum relative-errors for both biased and unbiased versions of the GreedyRel and IndDP algorithms as the average number of coefficient values that the IndDP algorithm uses per measure is varied from 10 to 60, and for the Mixed and AllNoPerm (in the specific order) selection of Zipfian distribution shapes (results for the AllNormal case are

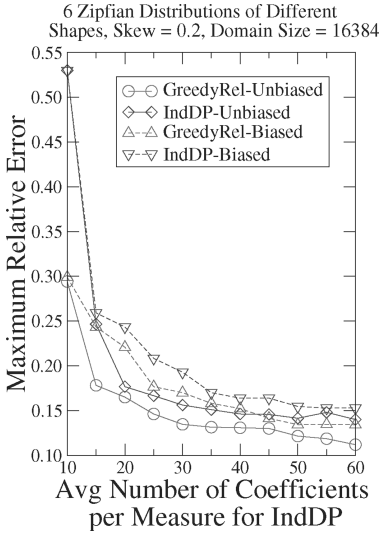

Fig. 19. Skew 0.2, Mixed.

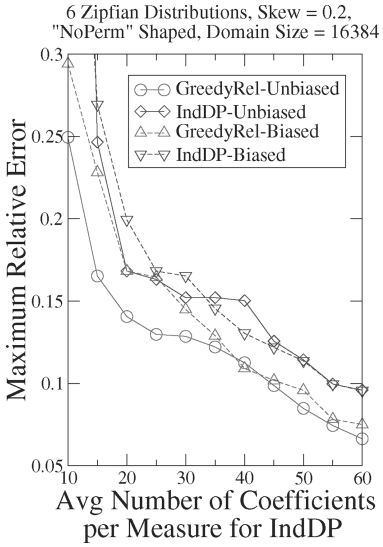

Fig. 20. Skew 0.2, AllNoPerm.

presented in the Electronic Appendix). As we can see, the GreedyRel produces more accurate results than the IndDP algorithm (direct comparisons should be made only between pairs of algorithms that use the same minimization method) for both biased and unbiased versions, with the differences being more significant in the AllNoPerm case (i.e., 25% versus 96% maximum relative-errors for the unbiased version and for a space constraint of ten coefficients per measure).

Similar results are also observed for the three combinations of synthetic datasets arising from setting the skew of the distributions to 0.8. For example,

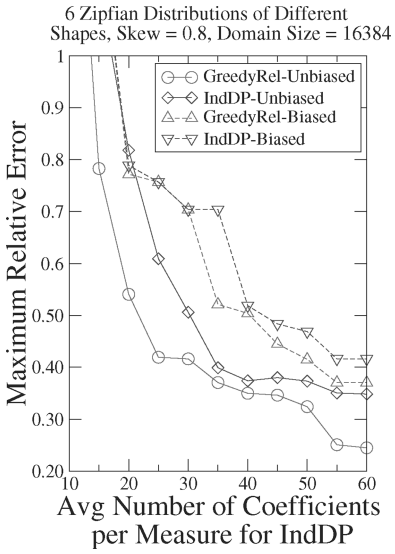

Fig. 21. Skew 0.8, Mixed.

in Figure 21 (results for the AllNoPerm and AllNormal combinations can be found in the Electronic Appendix), we show the corresponding results for the Mixed combination of used data distributions. In the unbiased case, we observe that the GreedyRel algorithm results in significantly smaller maximum relative-errors of the approximation than the IndDP algorithm in all cases. We need to note that in this set of experiments, the maximum relative-error achieved by all techniques for a space constraint equal to ten coefficient values per measures is larger than one. While we may argue that in such cases, an obvious "StoreNothing" solution that avoids storing any coefficient values would be preferable, since it would result in a maximum relative-error of one, this is not always the case. For example, in Figure 22 we depict the average relative-error achieved by our techniques as we vary the synopsis size for the Mixed dataset, with the skew parameter set to 0.8 (the largest average relative-errors were observed in this dataset). The benefits of the GreedyRel algorithm are significant, even in this case. Moreover, even for very small synopsis sizes, the GreedyRel algorithm achieves average relative-errors that are significantly lower than those of the aforementioned "StoreNothing" technique (which always exhibits a relative error of one for all nonzero values). This is more evident in other datasets, where the obtained average relative-error is significantly lower (i.e., in the AllNormal dataset in Figure 23).

Accuracy comparison of GreedyRel and IndDP in real datasets. In Figure 24 we plot the maximum relative-errors for a subset of the weather dataset, where the relative humidity, solar irradiance, air temperature, and wind peak measures have been extracted, as we vary the size of the synopsis, and for domain sizes of 16,384. As we can see, the benefits of the GreedyRel algorithm continue to be significant in all cases. While the IndDP algorithm fails to provide maximum relative-errors lower than 80%, the GreedyRel algorithm significantly tightens

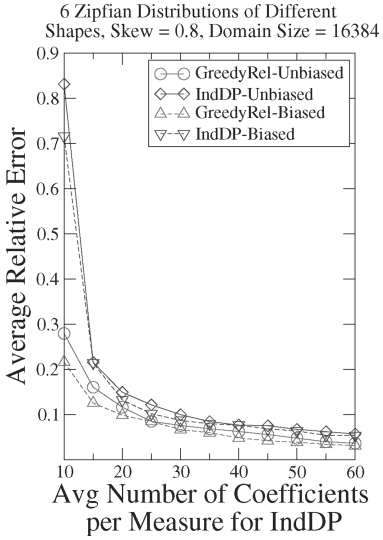

Fig. 22. Skew 0.8, Mixed.

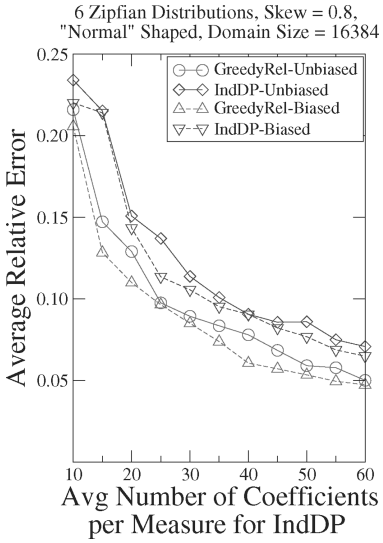

Fig. 23. Skew 0.8, AllNormal.

its error guarantees as the synopsis size increases. In the weather dataset, the GreedyRel algorithm provided up to five times tighter error bounds than the IndDP algorithm (and commonly, at least a two-fold improvement). However, similarly to the synthetic datasets, the average relative-errors were significantly smaller. For example, for the biased versions and for a synopsis space of 60KB, the GreedyRel algorithm resulted in an average relative-error of 1.6%, while the IndDP algorithm had a corresponding error of 3.6%.

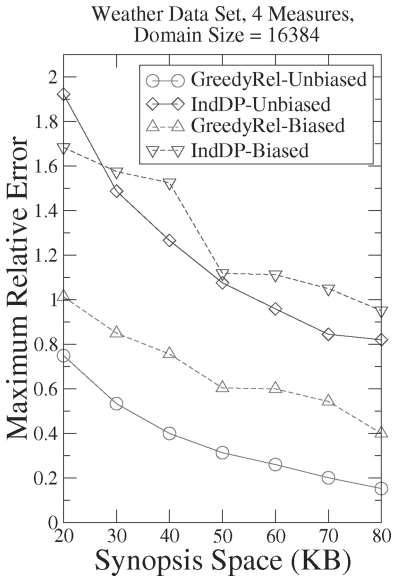

Fig. 24. Weather data.

6. RELATED WORK

Wavelet-based summarization techniques have been applied successfully in selectivity estimation [Matias et al. 1998], and approximately answering range-sum aggregates [Vitter and Wang 1999] as well as complex relational queries [Chakrabarti et al. 2001] over data cubes. Matias et al. [2000] consider the problem of online maintenance for large wavelet coefficients and propose a probabilistic counting scheme. More recently, Matias and Urieli [2005] proposed optimal algorithms for workload-based wavelet synopses that minimize weighted sum-squared (absolute or relative) error.

Guha et al. [2004] improve on the running time complexity of our DynProgL2 algorithm (originally in Deligiannakis and Roussopoulos [2003a]) to $O(NM(\log M + \log \frac{B}{S+\frac{B}{M}}) + \frac{M^2+B^2}{S+\frac{B}{M}})$, with a space complexity of $O(B^2+MB)$. One of their key observations is that the optimal solution needs to consider only a subset of candidate sets (i.e., ones that rank high in terms of benefit) amongst all candidate sets with equal numbers of stored coefficient values from all coefficients. Utilizing the same per-space benefit criterion for candidate sets, Guha et al. [2004] show that only a small number of candidate sets per combined coefficient need be considered, and that each candidate set needs to be considered for insertion in the final solution *only once* (instead of multiple times, as in our GreedyL2 algorithm). This observation, along with relaxation of the storage constraint, helps bound the size of their min-heap structure to $O(B)$ and improve the running time of their algorithm to $O(NM(\log M + \log B))$. Furthermore, these improvements also enable their algorithm to adapt to data streaming environments.

All of the aforementioned papers rely on conventional, L_2-error-based thresholding schemes that typically decide the significance of a coefficient based on

its absolute normalized value. Garofalakis and Gibbons [2004] were the first to identify the potential problems of such synopses and propose probabilistic thresholding algorithms specifically targeting the maximum relative-error. Building on this work, Garofalakis and Kumar [2005] introduce efficient deterministic thresholding schemes for general error metrics based on novel DP formulations. Recently, Karras and Mamoulis [2005] have proposed greedy deterministic thresholding heuristics for optimizing maximum-error metrics in one-dimensional datasets. Extending this work to multidimensional and/or multimeasure datasets is an open research problem, and could potentially provide effective deterministic solutions for the problems addressed in this article (Section 4.5). Finally, Guha [2005] presents a general methodology for reducing the space requirements of synopsis construction algorithms, essentially ensuring that the memory needed by a dynamic program never exceeds the size of its "working set;" of course, such techniques are generally applicable to the DP schemes presented in this article.

7. CONCLUSIONS

In this article, we introduced the notion of an extended wavelet coefficient as a flexible storage method for maintaining wavelet coefficients for datasets containing multiple measures. This flexible storage method bridges the gap between the two extreme storage hypotheses that the existing Individual and Combined algorithms represent, and achieves better storage utilization, which results in improved accuracy to queries. We also proposed a novel algorithm for constructing effective (optimal or near-optimal) extended wavelet-coefficient synopses that seek to minimize, given a storage constraint, either the sum-squared or maximum relative-error of the approximation. An extensive experimental study with both synthetic and real-life datasets validated our approach, demonstrating that extended wavelet synopses consistently outperformed existing techniques in the accuracy of the resulting approximation.

REFERENCES

ACHARYA, S., GIBBONS, P. B., POOSALA, V., AND RAMASWAMY, S. 1999. Join synopses for approximate query answering. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 275–286.

CHAKRABARTI, K., GAROFALAKIS, M., RASTOGI, R., AND SHIM, K. 2001. Approximate query processing using wavelets. *VLDB J. 10*, 2-3 (Sept.), 199–223.

CISCO. 1999. Netflow services and applications. White paper. `http://www.cisco.com/`.

COCHRAN, W. G. 1977. *Sampling Techniques*. John Wiley, New York.

CORMEN, T., LEISERSON, C., AND RIVEST, R. 1990. *Introduction to Algorithms*. MIT Press, Cambridge, MA.

DELIGIANNAKIS, A. AND ROUSSOPOULOS, N. 2003a. Extended wavelets for multiple measures. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 229–240.

DELIGIANNAKIS, A. AND ROUSSOPOULOS, N. 2003b. Extended wavelets for multiple measures. Tech. Rep. CS-TR-4462, University of Maryland.

FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. 1990. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA.

GANGULY, S., HASAN, W., AND KRISHNAMURTHY, R. 1992. Query optimization for parallel execution. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 9–18.

GAROFALAKIS, M. AND GIBBONS, P. B. 2001. Approximate query processing: Taming the terabytes. *Proceedings of the 27th International Conference on Very Large Data Bases*.

GAROFALAKIS, M. AND GIBBONS, P. B. 2004. Probabilistic wavelet synopses. *ACM Trans. Database Syst. 29*, 43–90.

GAROFALAKIS, M. AND KUMAR, A. 2005. Wavelet synopses for general error metrics. *ACM Trans. Database Syst. 30*, 279–332.

GILBERT, A., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. 2001. Surfing wavelets on streams: One-Pass summaries for approximate aggregate queries. In *Proceedings of the 27th International Conference on Very Large Data Bases*. 79–88.

GUHA, S. 2005. Space efficiency in synopsis construction algorithms. In *Proceedings of the 31st International Conference on Very Large Data Bases*. 409–420.

GUHA, S., KIM, C., AND SHIM, K. 2004. XWAVE: Approximate extended wavelets for streaming data. In *Proceedings of the 30th International Conference on Very Large Data Bases*. 288–299.

HELLERSTEIN, J., HAAS, P., AND WANG, H. 1997. Online aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 171–182.

JAWERTH, B. AND SWELDENS, W. 1994. An overview of wavelet based multiresolution analyses. *SIAM Rev. 36*, 377–412.

KARRAS, P. AND MAMOULIS, N. 2005. One-Pass wavelet synopses for maximum-error metrics. In *Proceedings of the 31st International Conference on Very Large Data Bases*.

MATIAS, Y. AND URIELI, D. 2005. Optimal workload-based weighted wavelet synopses. In *Proceedings of the 13th International Conference on Database Theory (ICDT)*. 368–382.

MATIAS, Y., VITTER, J., AND WANG, M. 2000. Dynamic maintenance of wavelet-based histograms. In *Proceedings of the 26th International Conference on Very Large Data Bases*. 101–110.

MATIAS, Y., VITTER, J. S., AND WANG, M. 1998. Wavelet-based histograms for selectivity estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 448–459.

SCHMIDT, R. R. AND SHAHABI, C. 2002. Propolyne: A fast wavelet-based algorithm for progressive evaluation of polynomial range-sum queries. In *Proceedings of the 8th International Conference on Extending Database Technology (EDBT)*. 664–681.

STOLLNITZ, E. J., DEROSE, T. D., AND SALESIN, D. H. 1996. *Wavelets for Computer Graphics - Theory and Applications*. Morgan Kaufmann, San Fransisco, CA.

VAZIRANI, V. V. 2001. *Approximation Algorithms*. Springer Verlag.

VITTER, J. AND WANG, M. 1999. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 193–204.

VITTER, J. S. 1985. Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*, 37–57.

Received September 2005; revised xxxx; accepted August 2006