Workload-Based Wavelet Synopses

Yossi Matias Leon Portman
School of Computer Science
Tel Aviv University
{matias,leonpo}@cs.tau.ac.il

(Dec 2003; rev. Sept. 2005)

Abstract

This paper introduces workload-based wavelet synopses, which exploit query workload information to significantly boost accuracy in approximate query processing. We show that wavelet synopses can adapt effectively to workload information, and that they have significant advantages over previous approaches. An important aspect of our approach is optimizing synopses constructions toward error metrics defined by workload information, rather than based on some uniform metrics. We present an adaptive greedy algorithm which is simple and efficient. It is run-time competitive to previous, non-workload based algorithms, and constructs workload-based wavelet synopses that are significantly more accurate than previous synopses. The algorithm also obtains improved accuracy for non-workload case when the error metric is the mean relative error.

We also present a self-tuning algorithm that adapts the workload-based synopses to changes in the workload. All algorithms are extended to workload-based multidimensional wavelet synopses with improved performance over previous algorithms. Experimental results demonstrate the effectiveness of workload-based wavelet synopses for different types of data sets and query workloads, and show significant improvement in accuracy even with very small training sets.

1 Introduction

In recent years there has been increasing attention to the development and study of data synopses, as effective means to address performance issues in massive data sets. Data synopses are concise representations of data sets, that are meant to effectively support approximate queries to the represented data sets [14]. A primary constraint of a data synopsis is its size, and its effectiveness is measured by the accuracy of the answers it provides, and by its build time and response time. Several different synopses were introduced and studied, including random samples, sketches, and different types of histograms. Recently, wavelet-based synopses were introduced and were shown to be effective data synopses for various applications.

In this paper, we introduce *workload-based wavelet synopses*, which exploit available query workload information to significantly boost accuracy over previous wavelet synopses, in answering range-sum and point queries. We present algorithms that are run-time competitive in performance to previous algorithms, which do not exploit the workload information.

The main algorithm is an adaptive-greedy algorithm which works for range-sum and point queries. Its adaptive nature turns out to be effective even for cases in which no workload is available.

We present two modification with improved run-time performance, using modified heuristics and an approximate data structure. The algorithm and its modifications are extended to a self-tuning algorithm that effectively addresses updates in the query workload. The algorithms are also extended to provide workload-based multidimensional wavelet synopses with improved performance over previous algorithms.

1.1 Related work

There has been extensive research in data synopses, including random samples, various histograms, and wavelet synopses.

Precomputed samples are based on the use of random samples as synopses for large data sets (see, e.g., [13, 3, 1, 2, 8, 7] and references therein).

Histograms are commonly used in database applications to capture the distribution of the data stored in a database relation, and are used to guide selectivity estimation as well as approximate query processing. Many different histograms have been proposed in the literature and some have been deployed in commercial RDBMSs. Some examples are equi-depth histograms (e.g., [1]), compressed histograms, v-optimal histograms, and maxdiff histograms; see the taxonomy given in [25].

Recently, *wavelet-based histograms* were introduced as a means for improved histogram accuracy [21, 30]. Wavelets are a mathematical tool for hierarchical decomposition of functions, whose adaptive nature make them good candidate for a "lossy" data representation. Wavelets represent functions in terms of a coarse overall shape, plus details that range from broad to narrow [27], thus offering an elegant technique for representing the various levels of function details in a space-efficient manner. Wavelet-based histograms exhibit significant improvement over basic histograms by adapting their construction to the nature of the underlying data sets. The use of wavelet-based synopses in databases has drawn increasing attention, with recent works on OLAP applications [29, 28], approximate query processing [6], probabilistic wavelet synopses [11], and extensions to multiple measures [9].

While the above works deal primarily with building data synopses for given data sets, they do not consider the query distribution with which the synopses will be used. In typical real-world scenarios, queries are heavily biased [10, 1, 5]. There have been several recent works that deal with workload-based samples and basic histograms [1, 5]. As argued in [8], identifying an appropriate precomputed sample that avoids large errors for an *arbitrary* query is virtually impossible. To address this problem, recent works have proposed using the *query workload* to guide the process of selecting samples [1, 7, 10]. The objective is to select a sample that will ensure acceptable errors for queries taken from a distribution represented by the query workload. Chaudhuri et al [8] formulate the problem of precomputing a sample as an optimization problem, whose goal is to select a sample that minimizes the error for the given workload. Babcock et al [4] argue that for many aggregation queries, appropriately constructed biased (non-uniform) samples can provide more accurate approximations than a uniform sample. They also point out that optimal type of bias varies from query to query.

Thus, there were two main lines of research closely related to this work. The first involves various wavelet synopses without consideration to the workload, and the second involves basic histograms and precomputed samples that are workload-aware. While wavelet-based histograms exhibit significant improvement over basic histograms, none of the prior works considered exploiting their adaptive nature of wavelets in the presence of workload information. In previous wavelet-based methods, coefficients are selected to be included in the synopses so as to optimize an overall error defined with the assumption that all queries have an equal probability to appear.

In this work, we extend and advance previous works on data synopses by combining the two previously independent research directions of workload-based sampling and wavelet-based synopses. We define workload-based error metrics and present effective algorithms for the construction of workload-based wavelet synopses that are tuned towards minimizing the workload-based error.

1.2 Contributions

In this paper, we present a novel approach to building wavelet-based synopses, exploiting workload information. The aim is to use the available memory most effectively for a given usage pattern. We exploit the adaptive nature of wavelets to adapt the synopses not only to the given data, but also to the estimated probability distribution from which queries are taken, as it is represented by the query workload. We show that this can significantly improve the accuracy for queries drawn from a similar workload distribution.

The main contributions of this paper are:

- The approach taken in this paper is a departure from earlier works on wavelet-based synopses in terms of the way it deals with error metrics. Previous approaches optimize synopses rigorously towards minimizing the error with respect to an error metric defined for the uniform query distribution. Still, such synopses were tested using non-uniform test sets, as is appropriate for real life data sets. In contrast, we define *workload-based error metrics* and use them for the *construction* of wavelet-based

synopses, and for their maintenance. For instance, as is well known, conventional wavelet synopses that retain wavelet coefficients with the largest absolute value after normalization according to the level minimize the overall L^2 error in reconstructing all data values. In contrast, for a workload set of queries that is far from uniform, the performance of such synopses is far from optimal. As demonstrated, wavelets are particularly useful in adapting to error metrics as defined in an ad-hoc fashion by the workload.

- We present a novel data structure, called the *workload-matrix*, that is simple, yet powerful, and which enables efficient computations and updates of error contributions for wavelet coefficients, with respect to the workload queries.

- The new workload-based algorithms presented here may improve performance even for *uniform* workload, relative to the basic and the probabilistic algorithms, with respect to the mean-relative error (MRE), as demonstrated by experiments.

Wavelet synopses are typically constructed by computing the wavelet transform of the given data set, and then selecting a subset of the coefficients that are retained as a synopses. The selection process is called *thresholding*, and different wavelet synopses are based on different thresholding methods. The following novel approaches and algorithms are used within our workload-based thresholding:

- An *adaptive greedy* algorithm: We propose a workload-based, efficient algorithm for constructing wavelet-based synopses. It uses the workload matrix to exploit the workload information for calculation of the error contribution for wavelet coefficients in the thresholding process. Given a data set consisting of N distinct values and a training workload of size w, the algorithm requires $O(N + w)$ space and $O((N + w \log^2 N) \log N)$ construction time. For any range or point query, an approximate answer can be given in $O(\log N)$ time.

- A *backward thresholding* algorithm: We present and analyze a backward thresholding modification of our first algorithm, when instead of starting with a full set of candidates and exclusion of the candidates for thresholding, we start with the empty set and select candidates to keep in the synopses. The complexity of this algorithm is $O(N + (w + m + \frac{mw \log N}{N}) \log N)$ time with $O(w + N)$ space, where m is a number of retained coefficients.

- A *linear time approximate greedy* algorithm: We modify our adaptive greedy algorithm to achieve linear running time by employing an approximate priority queue without noticeable degradation in accuracy. The complexity of this algorithm is $O(N + w \log^2 N)$ time and still with $O(N + w)$ space.

We also extend our algorithms to self-tuning and multidimensional cases:

- A *self-tuning* algorithm: We present an algorithm for efficient self-tuning of wavelet-based synopses to updates in the workload. We demonstrate how this algorithm can effectively adapt and capture ad-hoc changes in the workload.

- A *multidimensional-wavelet synopsis* algorithm: We extend the definitions of workload to the multidimensional case and modify the algorithms in a rather simple way to construct workload-based multidimensional wavelet synopses.

We demonstrate the effectiveness of our methods in providing fast, highly accurate answers for given fixed and dynamic workloads, using real-world and synthetic data sets. Our workload-based algorithms achieve better error measures than non workload-based algorithms for different types of workloads. For instance, our experimental results for real-word data sets show that with 50 retained coefficients we achieve a 30 times improvement in the mean relative error over the basic thresholding algorithm.

1.3 Paper Outline

The rest of this paper is structured as follows. In Section 2 we describe the background material on wavelet synopses and approximate query processing. In Section 3 we introduce details on the workload-based wavelet

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

Table 1: Haar Wavelet Decomposition

thresholding concepts and define new workload-based error metrics. Section 4 presents our main algorithms and complexity analysis. Section 5 describes the self-tuning case for the changes in the workloads. In Section 6 we present an extension to our basic algorithms, dealing with the multi-dimensional case. Experimental results are presented in Section 7, followed by our conclusions.

2 Wavelet Basics

Wavelets are a mathematical tool for the hierarchical decomposition of functions in a space-efficient manner. Wavelets represent a function in terms of a coarse overall shape, in addition to details ranging from coarse to fine.

In this section we will start by presenting the Haar wavelets and continue by presenting wavelet based synopses, obtained by thresholding processes.

2.1 One-dimensional Haar wavelets

Haar wavelets are conceptually the simplest wavelet basis functions, and as with previous works, we focus on them for the purpose of exposition in this paper. They are fast to compute and easy to implement. To illustrate how Haar wavelets work, we start with a simple example borrowed from [21, 30]. Suppose we have a one-dimensional "signal" of $N = 8$ data items:

$$S = [2, 2, 0, 2, 3, 5, 4, 4]$$

We will start with performing a wavelet transform on it. We first average the signal values, pairwise, to get the new lower-resolution signal with values

$$[2, 1, 4, 4]$$

That is, the first two values in the original signal (2 and 2) average to 2, and the second two values 0 and 2 average to 1, and so on. Clearly, some information is lost in this averaging process. To recover the original signal from the four averaged values, we need to store some *detail coefficients*, which capture the missing information. Haar wavelets store the pairwise differences of the original values (divided by 2) as detail coefficients. In the above example, the four detail coefficients are $(2 - 2)/2 = 0$, $(0 - 2)/2 = -1$, $(3 - 5)/2 = -1$, and $(4 - 4)/2 = 0$. It is easy to see that the original values can be recovered from the averages and differences.

We have succeeded in the decomposition of the original signal into a lower-resolution version of half the numbers of the entries and a corresponding set of detail coefficients. By repeating this process recursively on the averages, we get the full decomposition (Table 1). We define the *wavelet transform* (also called *wavelet decomposition*) of the original eighth-value signal to be the single coefficient representing the overall average of the original signal, followed by the detail coefficients in the order of increasing resolution. Thus, for the one-dimensional Haar basis, the wavelet transform of our signal is given by

$$\widetilde{S} = [2\frac{3}{4}, -1\frac{1}{4}, \frac{1}{2}, 0, 0, -1, -1, 0].$$

The individual entries are called the *wavelet coefficients*. The wavelet decomposition is very efficient computationally, requiring only $O(N)$ CPU time and $O(N/B)$ I/Os to compute for a signal of N values, where B is the disk-block size.

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

2.2 Thresholding

Given a limited amount of storage for maintaining a wavelet synopsis of a data array A, we can only retain a certain number m of the coefficients stored in the wavelet decomposition of A. The remaining coefficients are implicitly set to 0. The goal of coefficient thresholding is to determine the "best" subset of m coefficients to retain, so that some overall error measure in the approximation is minimized.

One advantage of the wavelet transform is that in many cases a large number of the detail coefficients turn out to be very small in magnitude. Truncating these small coefficients from the representation (i.e., replacing each one by 0) introduces only small errors in the reconstructed signal. We can approximate the original signal effectively by keeping only the most significant coefficients.

For a given input sequence $d_1, d_2, \ldots, d_N$, we can measure the error of the approximation in several ways. We focus on range queries $q(l:h)$, whose result is $d(l:h) = \sum_{i=l}^{h} d_i$; let $q(j) = q(j:j)$ the result of which is d_j. Let $\hat{d}(l:h)$ be the estimated result of the query $q(l:h)$. We use the following error measure for the *relative error* $e(q)$ of query $q = q(l:h)$:

$$e(q(l:h)) = \frac{d(l:h) - \hat{d}(l:h)}{d(l:h)}, \text{ for } d(l:h) \neq 0.$$

Once we have the error measure for representing the errors of individual queries, we need to choose a norm by which to measure the overall error of a collection of w queries $Q = \{q_1, q_2, \ldots, q_w\}$. We define the following error measures:

1. The *mean square error*:

$$\text{MSE}(Q) = \frac{1}{w} \sum_{j=1..w} (d(l:h) - \hat{d}(l:h))^2$$

2. The *mean relative error*:

$$\text{MRE}(Q) = \frac{1}{w} \sum_{j=1..w} |e(q_j)|$$

3. The *max relative error*:

$$\text{MAXRE}(Q) = max_j\{e(q_j)\}$$

The max relative error helps in the assessment of the amount of bias in the approximation. Such an error can be used to provide guarantees on the quality of approximate results for individual queries.

The basic thresholding algorithm [27] is as follows: let $a_0, ..., a_{N-1}$ be the detail coefficients of the wavelet decomposition, and for each a_i let $level(a_i)$ be the level of resolution at which the coefficient appears. The detail coefficients are normalized by dividing each coefficient by $\sqrt{2^{level(a_i)}}$ reflecting the fact that coefficients at the lower resolutions are "less important" than the coefficients at the higher resolutions. The M highest normalized coefficients are retained. The remaining $N - M$ coefficients are implicitly replaced by zero. This deterministic process is known [27] to *provably* minimize the mean-square error, i.e., the L_2 norm of the vector of errors defined above.

2.3 Error tree

The wavelet decomposition procedure followed by any thresholding can be represented by an *error tree* [21]. Figure 1 presents the error tree for the above example. Each internal node of the error tree is associated with a wavelet coefficient, and each leaf is associated with an original signal value. Internal nodes and leaves are labeled separately by $0, 1, ...N-1$. For example, the root is an internal node with label 0 and its node value is 2.75 in Figure 1. For convenience, we shall use "node" and "node value" interchangeably.

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

2.4 Reconstruction of the original data

Given an error tree T and an internal node t of T, $t \neq c_0$, we let $leftleaves(t)$ ($rightleaves(t)$) denote the set of leaves (i.e., data) nodes in the subtree rooted at t's left (resp., right) child. Also, given any (internal

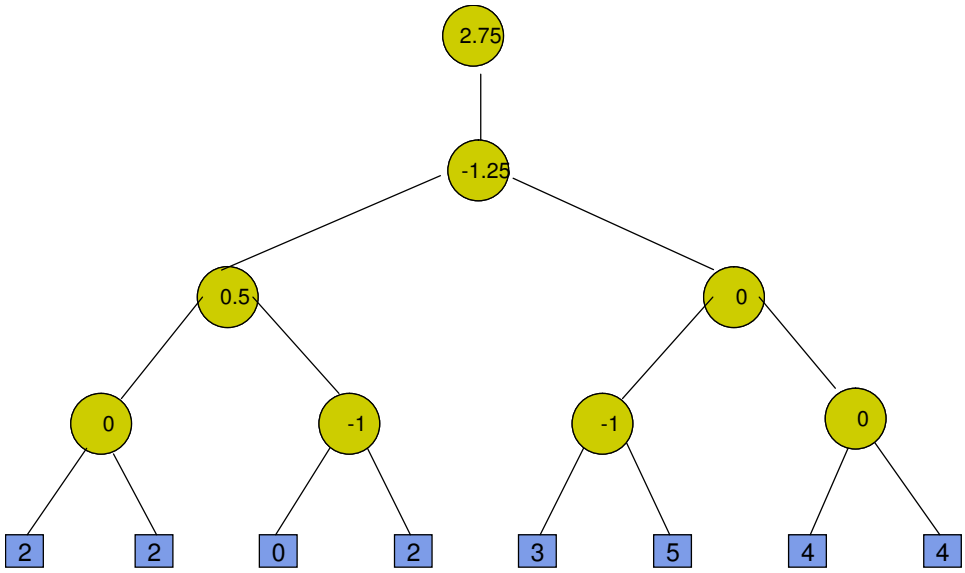

Figure 1: Error tree for $N = 8$

or leaf) node u, we let $path(u)$ be the set of all (internal) nodes in T that are proper ancestors of u (i.e., the nodes on the path from u to the root of T, including the root but not u) with non-zero coefficients. Finally, for any two leaf nodes d_l and d_h we denote $d(l:h)$ as the range sum $\sum_{i=l}^{h} d_i$. Using the error tree representation T, we can outline the following reconstruction properties of the Haar wavelet decomposition [21, 30]:

Single value. The reconstruction of any data value d_i depends only on the values of the nodes in $path(d_i)$.

$$d_i = \sum_{c_j \in path(d_i)} \delta_{ij} \cdot c_j \tag{1}$$

where $\delta_{ij} = +1$ if $d_i \in leftleaves(c_j)$ or $j = 0$, and $\delta_{ij} = -1$ otherwise.

Range sum. An internal node c_j contributes to the range sum $d(l:h)$ only if $c_j \in path(d_l) \cup path(d_h)$.

$$d(l:h) = \sum_{c_j \in path(d_l) \cup path(d_h)} x_j \tag{2}$$

where

$$x_j = \begin{cases} (h-l) \cdot c_j & \text{if } j = 0 \\ (|leftleaves(c_j, l:h)| - & \\ |rightleaves(c_j, l:h)|) \cdot c_j & \text{otherwise,} \end{cases}$$

and where $leftleaves(c_j, l:h) = leftleaves(c_j) \cap d_l, d_{l+1}, ..., d_h$ (i.e., the intersection of $leftleaves(c_j)$ with the summation range) and $rightleaves(c_j, l:h)$ is defined similarly.

Thus, a reconstruction of a single data values involves the summation of at most $\log N + 1$ coefficients, and reconstructing a range sum involves the summation of at most $2 \log N + 1$, coefficients, regardless of the width of the range.

3 Workload-based thresholding

In this section we describe the problems with previous wavelet synopses, regarding the error metric used in their construction and evaluation, by presenting a simple motivating example. We then present our general approach for defining workload-based error metrics for construction and evaluation of wavelet synopses. As a key ingredient of our algorithms, we define the *coefficient error contribution*, which enables us to assess the importance of wavelet coefficients during the thresholding process. We also describe the *workload matrix*, which will be used as a key data structure in the workload-based algorithms, presented in the next section.

1	index	1	2	3	4	5	6	7	8	MSE	MSE(W)
2	given data	2	2	2	6	**3**	**5**	**4**	**4**		
3	wavelet coeff.	$3\frac{1}{2}$	$-\frac{1}{2}$	-1	0	0	-2	-1	0		
4	norm. coeff.	$3\frac{1}{2}$	$-\frac{1}{4}$	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	$-\frac{1}{8}$	0		
5	basic retained	$3\frac{1}{2}$	$-\frac{1}{2}$	-1			-2				
6	work. retained	$3\frac{1}{2}$	$-\frac{1}{2}$	-1				-1			
7	basic approx.	2	2	2	6	**4**	**4**	**4**	**4**	0.019	0.038
8	work. approx.	2	2	4	4	**3**	**5**	**4**	**4**	0.139	0

Table 2: Basic and workload-based thresholding

3.1 A motivating example

Previous thresholding methods select coefficients based on their contribution in the context of a general error metric, which assumes that all queries are equally likely. The choice of coefficients can be far from optimal in cases where the actual query distribution is skewed. We illustrate this in the following simple example, depicted in Table 2 and Figures 2 and 3 and show how a choice of coefficients affected by the given set of queries could be significantly better.

Table 2 presents a sequence of 8 data items (second row), which are the leaves of Error Tree in Figure 2, followed by the corresponding sequence of wavelet coefficients (3rd row), and followed by their corresponding normalized values (4th row). Let's assume that we can retain $m = 4$ coefficients. The 5th row consists of the basic retained coefficients, which have the highest absolute normalized value: c_1, c_2, c_3, c_6, as depicted in Figure 2.

The 7th row consists of the results for all single value queries $\{q(i)\}$, as obtained using the basic retained coefficients. The L^2 error for these coefficients is provably optimal with respect to all possible choices of 4 retained coefficients, and can be calculated as: $L^2 = ((1/9) + (1/25))/8 \approx 0.019$.

Let the given workload, W, consist of queries that involve only the last four data values (d_5, d_6, d_7, d_8). For instance, let $W = \{q(5), q(6), q(7), q(8)\}$. The only results of interest for Q are shown in bold-face, and the overall error for W is $\text{MSE}(W) = ((1/9) + (1/25))/4 \approx 0.038$ as shown in Figure 2.

A workload-based selection of retained coefficients, which adapts to the given set W, will retain c_7 instead of c_3. The workload-based retained coefficients are given in the 6th row, and depicted in Figure 3. Similarly to row 7, the 8th row consists of the results for all single value queries, $\{q(i)\}$, as obtained using the workload-based retained coefficients. The L^2 error for the workload-based retained coefficients can be calculated as: $L^2 = (1 + (1/(9)/8) \approx 0.139$ (Figure 2). It is indeed significantly larger than the one obtained for the basic retained coefficients. However, our interest is in the overall error for the given set of queries W, for which we have $\text{MSE}(W) = 0$ since all relevant coefficients for the given queries are retained. This is significantly better than the error obtained for the basic method, in which $\text{MSE}(W) \approx 0.038$.

The problem with previous methods, as illustrated above, is that they optimize the retained coefficients according to some overall error measures, such as L^2, in which all queries are assumed to be equally likely. In contrast, the quality evaluation of the wavelet synopsis is defined according to the given query sets.

3.2 Workload-based error metrics

Our goal is to construct wavelet synopses that would minimize the error for a future set of queries Q. A *workload* W is a given set of queries that is considered to be a good representation of a probability distribution from which future queries are to be generated. The general approach is hence to construct a wavelet synopses that minimize the error with respect to the workload W.

Given a workload W that consists of w queries $q_1 ... q_w$, and a set C of retained coefficients, the *mean squared error for the workload W with respect to C* is defined as:

$$\text{MSE}(W, C) = \frac{1}{w} \sum_{j=1..w} e_{abs}(q_j, C)^2, \tag{3}$$

$$\text{MRE}(W, C) = \frac{1}{w} \sum_{j=1..w} |e(q_j, C)|, \tag{4}$$

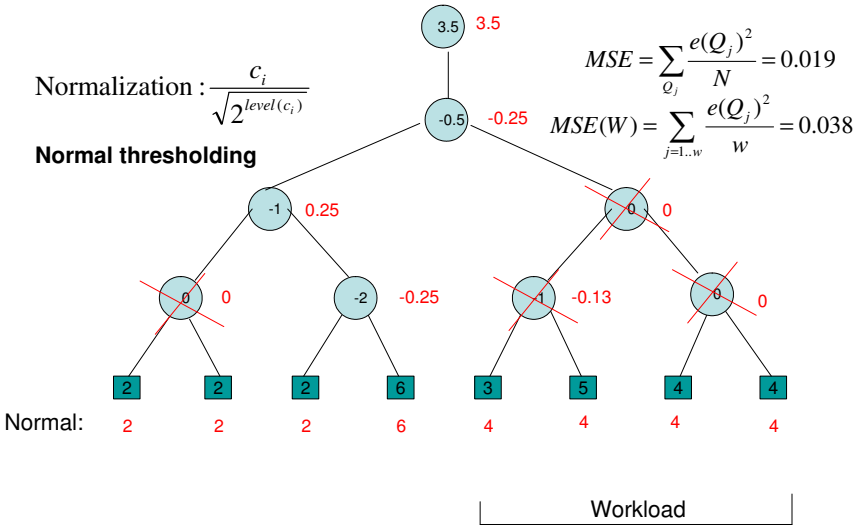

Figure 2: Motivating example, Step 1

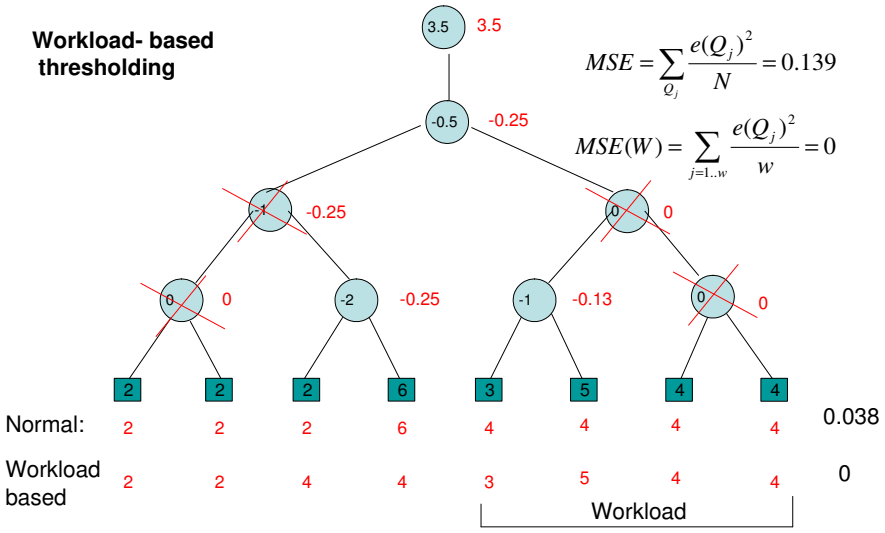

Figure 3: Motivating example, Step 2

$$\text{MAXRE}(W, C) = max_j\{e(q_j, C)\}, \tag{5}$$

where for every $q_j = q(l : h)$, we have

$$e(q_j, C) = \frac{d(l : h) - \hat{d}(l : h)}{d(l : h)}, \text{ for } d(l : h) \neq 0.$$

$$e_{abs}(q_j, C) = d(l : h) - \hat{d}(l : h)$$

and $\hat{d} = \hat{d}(C)$ is calculated using the retained coefficients in C. When the set C will be apparent from the context, we will use MSE(W), $e(q_j)$, and $\hat{d}$. Note that if W consists of all possible queries, then MSE(W) is the L^2-error.

3.3 Coefficient error contribution

A key component in our adaptive algorithms is the consideration of the impact of inclusion or exclusion of coefficients on the workload-based error. Given a (tentative) set C of retained coefficients, we define the *error contribution*, e_i, of a coefficient c_i from C as the increase in error as a result of removing c_i from C. In particular for the MSE(W) metric, we will define:

$$e_i = e_i(W, C) = \text{MSE}(W, C \setminus \{c_i\}) - \text{MSE}(W, C)\,.$$

The coefficient error contribution can be similarly defined for the MRE(W) and the MAXRE(W) metrics:

$$e_i = e_i(W, C) = \text{MRE}(W, C \setminus \{c_i\}) - \text{MRE}(W, C)\,.$$

$$e_i = e_i(W, C) = \text{MAXRE}(W, C \setminus \{c_i\}) - \text{MAXRE}(W, C)\,.$$

Similarly we define the error contribution of c_i to be the *decrease* in the error measure in case c_i is added to a set C of retained coefficients that does not include c_i.

3.4 Composite error metrics

To address situations in which the dependency of Q in W is limited, a refined method is used, in which a weighted combination of MSE(W) and L^2-error are minimized. We define the following *composite error metric*

$$\text{MSE}'(W, C) = \nu \cdot \text{MSE}(W, C) + (1 - \nu) \cdot \text{MSE}(C) \tag{6}$$

The weight factor ν reflects the significance of the workload W for the thresholding process. For very large ν, the situation is similar to the pure workload-based contribution. For very small ν, the situation is similar to non-workload based setting.

Similarly we define the composite contribution of a coefficient c_i:

$$e'_i = e_i(W, C, \nu) = \nu \cdot e_i(W, C) + (1 - \nu) \cdot e_i(C) = \nu \cdot e_i(W, C) + (1 - \nu) \cdot c_i / \sqrt{2^{level(c_i)}}$$

The composite error can be extended to multiple workloads. This represents cases in which there are different workloads from different sources, and a combination of usage from these sources is expected. Given a set of workloads $W_1 ... W_k$ and a set of weights $\nu_1 ... \nu_k$, such that $\sum_{i=1..k} \nu_i = 1$, we define the *composite error metric* as:

$$\text{MSE}'(W, C) = \sum_{i=1..k} \nu_i \cdot \text{MSE}(W_i, C) \tag{7}$$

Similarly, the *error contribution* of a coefficient c_i will be given as:

$$e'_i = \sum_{i=1..k} \nu_i \cdot e_i(W_i, C)$$

For simplicity the algorithms described in this paper will only deal with the pure workload-based error metric (Eq. 3, 4, 5).

	q_1	q_2	q_3	...	q_w
c_1	1	1	1	...	1
c_2	1	1	0	...	1
c_3	1	0	0	...	0
...	...	...	...	...	...
c_N	0	1	0	...	1
d_j	d_1	d_2	d_3	...	d_w
$\hat{d}_j$	$\hat{d}_1$	$\hat{d}_2$	$\hat{d}_3$	...	$\hat{d}_w$

Figure 4: Workload Wavelet Coefficients Matrix

3.5 The basic greedy heuristic

A natural approach to thresholding is utilizing a greedy heuristic, in a spirit similar to the standard basic thresholding described in Section 2.2. The synopsis consists of the coefficients whose error-contributions with respect to the workload W and to the set C consisting of all the wavelet coefficients. Since in this case $\text{MSE}(W, C) = 0$, the error contribution is $e_i = e_i(W, C) = \text{MSE}(W, C \setminus \{c_i\})$. A straightforward computation of all error contributions takes $O(Nw)$ time. However, a more careful implementation would take $O(w \log N)$ time, using Eq. 2 and observing that up to $2 \log N$ coefficients can contribute to an error of $q_j \in W$.

A general limitation of the greedy heuristic is that it is computed with respect to coefficient error contributions that are computed for a setting in which all the coefficients are part of the synopses. The hierarchical structure of these coefficients make the error contributions highly correlated. Thus, a removal of a coefficient from the synopsis could significantly change the error contributions of other coefficients. The only situation in which this is not the case is when the Haar basis is orthonormal with respect to the inner product defined by the error metric of interest, where Parseval's theorem implies that the greedy heuristic is optimal (see [20, 19] for an elaborated discussion). This is the case for the MSE metric and the uniform workload case, but is not the case for either other workloads or other error metrics.

In the next section we address this limitation by devising an adaptive algorithm in which the error contributions are adapted to intermediate selections. The constructed synopsis has improved accuracy over the greedy heuristic for both non-uniform workloads as well as for uniform workload with the MRE metric.

4 Adaptive thresholding algorithms

In this section we present the adaptive thresholding algorithms for workload-based wavelet synopses. The thresholding algorithm is based on iteratively removing from the set of candidate coefficients the one whose removal will induce the smallest increase in the workload-based error. After a coefficient is removed, the set of candidate coefficients C is changed, and the error contributions are updated accordingly. Thus, the algorithm is greedy in each selection, but is adaptive to the selections done so far. We first present the primary data structure – the workload matrix – which is based on the coefficient error contribution presented in the previous section. We then present the adaptive algorithm implementation, which uses the workload matrix.

4.1 The Workload matrix

A frequent operation in the adaptive thresholding algorithms is to obtain, for a given coefficient c_i under consideration of inclusion or exclusion, the subset of queries in W to which c_i contributes, and the subset of coefficients that also contribute to the same queries. For this purpose, we present the following *workload matrix* data structure.

The workload matrix, depicted in Figure 4, consists of N rows, one per wavelet coefficient, and w' columns, one per each distinct query in W ($w' \leq w$). For each c_i and q_j, the entry at cell (i, j) is "1" if c_i is used for the calculation of q_j, and is "0" otherwise. As will be seen next, the matrix is very sparse. It is implemented as a combination of hash tables and linked lists for every row and for every column, in space linear in the number of "1" entries. In addition, for each query q_j, we will keep its exact result, d_j, and a

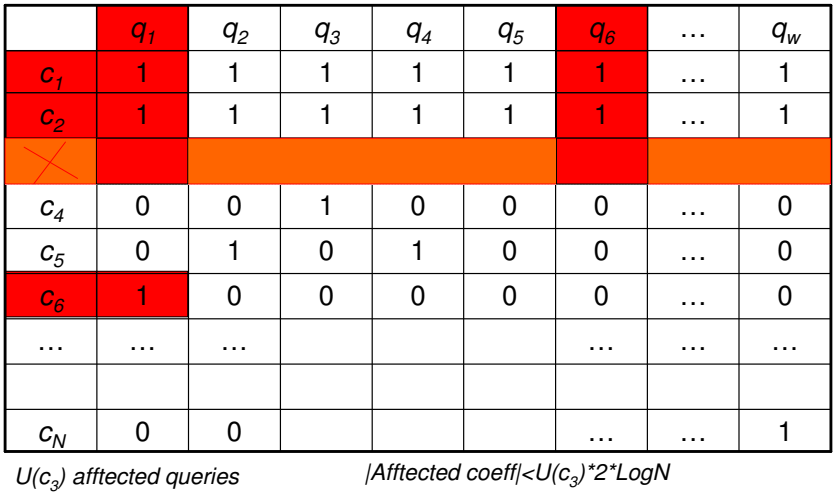

Figure 5: **Affected coefficients.** Coefficient c_3 contributes to the computation of queries q_1 and q_6, which are its affected queries. Coefficients c_1, c_2, and c_6 also contributes to the computation of queries q_1 and q_6, and they are therefore affected coefficients. Each affected query can be related to at most $2 \log N$ affected coefficients.

tentative approximate result $\hat{d}_j$, associated with the corresponding column. The workload matrix will be used by the thresholding algorithm, and is not part of the wavelet synopsis.

Let $U(c_i)$ be a set of queries that use some wavelet coefficient c_i Sec. 2.2 Note that $U(c_i)$ is the set of queries that correspond to the positions of '1' entries in the i'th row. Let $D(q_j)$ be the set of coefficients that are used in the computation of q_j Sec. 2.2. Note that $D(q_j)$ is the set of coefficients that correspond to the positions of '1' entries in the j'th column.

Lemma 1. *Let $\bar{U}$ be the average size of $U(c_i)$ over all wavelet coefficients: $\bar{U} = \frac{1}{N} \sum_{i=1..N} |U(c_i)|$.*

$$\bar{U} \leq \frac{w \cdot 2 \cdot \log N}{N} . \tag{8}$$

Proof. Clearly, according to the definition of the query processing Sec. 2.2, we have

$$|D(q_j)| \leq 2 \log N .$$

Hence, since $|D(q_j)|$ is the number of '1's in column j, there are at most $2 \log N$ "1"'s in each column and at most $w \cdot 2 \cdot \log N$ '1's in the entire workload matrix. The lemma follows by observing that $N\bar{U}$ is the number of '1's in the matrix. ∎

Composite error metric. The workload matrix data structure can be easily extended to support the composite error metric (Section 3.4) by using a separate workload matrix for each workload. The error contribution of a coefficient will be evaluated by executing a lookup to each of the workload matrices, and combining the results.

Weighted workload. The workload matrix can support a *weighted workload*, where for every query q_j in W, there is a weight factor σ_j. The above definitions and calculations can be adapted to such setting. Due to the additivity of the error contribution, occurrences of $e(q_j)$ can be essentially replaced by $e(q_j) \cdot \sigma_j$. The workload matrix is extended so that in every non-zero entry (i, j), the "1" will be replaced by σ_j. Note that this is also relevant to a case in which W is a multiset, in which case, the weight of a query will represent its count. The weighted workload matrix is depicted in Figure 6.

	q_1	q_2	q_3	...	q_w
c_1	σ_1	σ_2	σ_3	...	σ_w
c_2	σ_1	σ_2	0	...	σ_w
c_3	σ_1	0	0	...	0
...	...	...	...	...	...
c_N	0	σ_2	0	...	σ_w

Figure 6: The weighted workload wavelet coefficients matrix

AdaptiveThreshold(C, m, W)

Build the workload matrix data structure
for each $c_i \in C$
  do calculate its (initial) error contribution (e_i)
Build a priority queue PQ for the coefficients, based on e_i
while $|C| > m$
  do
    Extract the top coefficient (c_i) from the PQ
    $C = C \setminus \{c_i\}$
    for each $q_j \in U(c_i)$
      do Update approximate results $(\hat{d}_j)$
    for each $q_j \in U(c_i)$
      do
        for each $c_{i'} \in D(q_j) \bigcap$PQ
          do Update the error contributions $(e_{i'})$

Figure 7: The adaptive thresholding algorithm

The weighted workload matrix retains all the previously presented properties, where instead of occurrences of '1's, we count the occurrences of non-zero σ_js, and w represents the number of distinct queries in the workload.

4.2 The adaptive algorithm implementation

The input of the algorithm consist of: (1) an array $C = [c_1, ..., c_N]$ of N Haar wavelet coefficients; (2) m – the number of coefficients to keep; and (3) W – the training workload that consists of w queries. The output of the algorithm is an array $C' = [c_{i_1}, ..., c_{i_m}]$ of m non-zero retained coefficients. We use the workload matrix and a priority queue where the priorities are based on the error contributions.

In the first step, the algorithm builds the workload matrix data structure from the given wavelet coefficients and the workload. Then the algorithm calculates the initial error contribution for each coefficient in the workload matrix, builds the priority queue of the coefficients, based on the calculated initial error contribution. The main iteration of the algorithm is:

Step 1 Select the coefficient with the smallest error contribution from the priority queue.

Step 2 Threshold it and update approximation results of all the updated queries in the workload matrix.

Step 3 For each updated query, update the error contributions of the query's coefficients.

Step 4 Update the priority queue.

When the necessary number of coefficients is thresholded, the algorithm returns the remaining coefficients. A pseudo-code of the algorithm is depicted in Figure 7.

Theorem 1. *The Adaptive Workload-Based Wavelet Thresholding algorithm runs in* $O((N+w\log^2 N)\log N)$ *time and* $O(N+w)$ *space.*

Proof. We show the proof for the MRE; a similar proof applies for the other error metrics First we show how we can effectively calculate the error contributions (e_i) of the wavelet coefficients (c_i) at each thresholding iteration of the algorithm. Next, we show how to update the error contributions ($e_{i'}$) of the affected coefficients.

Let C^k be the set of retained coefficients at the beginning of iteration k. The error contribution of excluding some coefficient c_i from C is calculated as follows.

$$e_i = e_i(W, C^k) = \mathrm{MRE}(W, C^k) - \mathrm{MRE}(W, C^k \setminus \{c_i\})\,.$$

By substituting MSE using Eq. 4, we get

$$e_i = \sum_{j=1..w} \frac{e(q_j, C^k)^2}{w} - \sum_{j=1..w} \frac{e(q_j, C^k \setminus \{c_i\})^2}{w} =$$

$$= \sum_{j=1..w} \frac{e(q_j, C^k)^2 - e(q_j, C^k \setminus \{c_i\})^2}{w}\,.$$

By definition, if $q_j \notin U(c_i)$ then the computation of q_j does not involve c_i, and therefore $e(q_j, C^k)^2 = e(q_j, C^k \setminus \{c_i\})^2$. Hence,

$$e_i = \frac{1}{w} \sum_{q_j \in U(c_i)} \Big(\frac{d_j - \hat{d}_j(C^k)}{d_j}\Big)^2 - \Big(\frac{d_j - \hat{d}_j(C^k \setminus \{c_i\})}{d_j}\Big)^2$$

By using Eq. 2 we can write:

$$\hat{d}_j(C^k \setminus \{c_i\}) = \sum_{c_j \in path(d_l(q_j)) \cup path(d_h(q_j))} x_j = \hat{d}_j(C^k) - x_i\,.$$

For the initial error contribution we will have:

$$\hat{d}_j(C^0) = d_j \qquad \hat{d}_j(C^1) = d_j - x_i\,.$$

Thus, updating an approximate result $\hat{d}_j$, as a result of the thresholding of a coefficient c_i, such that $q_j \in U(c_i)$, takes $O(1)$ time. Furthermore, whenever a coefficient c_i is thresholded, all affected results $\hat{d}_j$ can be updated in $O(|U(c_i)|)$ time. By keeping all approximate results $\hat{d}_j$ up to date for the updated set of candidate coefficients C, the error contribution $e_{i'}$ of any (affected) coefficient $c_{i'}$ can be computed in $O(|U(c_{i'})|)$ time.

To see more closely which coefficients will actually be affected by the thresholding of c_i, we refer to the "error tree" structure (Figure 1) from Section 2.3. Suppose that the kth step of the greedy thresholding excludes a coefficient that corresponds to the node c_i in the error tree. The affected nodes, whose error contributions need to be updated, fall into two classes (Figure 8): (a) the wavelet coefficients in the subtree rooted at c_i; and (b) the wavelet coefficients on the path from c_i up to the root of the error tree. Each query affects at most $2\log N$ coefficients, so the average number of coefficients that need to be updated is at most $2\bar{U}\log N$; it is $|U(c_i)|\log N$ in the worst case.

In order to bound the number of updates, we observe that for each affected coefficient $c_{i'}$, even if $U(c_{i'})$ is larger than $U(c_i)$, we could still update its error contribution $e_{i'}$ in $O(|U(c_i)|)$ time, based on the fact that the number of queries $q_{j'} \in U(c_{i'})$ for which $\hat{d}_{j'}$ has been changed is bounded by $|U(c_i)|$. The actual update is done by iterating the list of row i in the workload matrix, and for each j such that entry (i,j) is '1' ($q_j \in U(c_i)$), traversing the list of column j and for each entry (i',j) that is '1' ($q_j \in U(c_{i'})$), updating $e_{i'}$ by adding the appropriate difference contributed by $\hat{d}_j(C^k \setminus \{c_i\})$.

In total, the number of updates for a single thresholding step of coefficient c_i is at most $2|U(c_i)|\log N$ in the worst case, with $O(1)$ cost per update. The total cost of thresholding all coefficients is at most $\sum_{i=1}^{N} 2|U(c_i)|\log N = 4w\log N \cdot \log N$, based on Lemma 1.

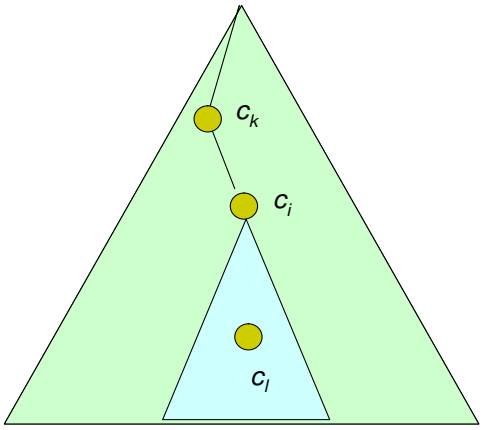

Figure 8: Affected coefficients when thresholding c_j

The priority queue of threshold candidates. The selection of candidates for thresholding is done using a priority queue. Each coefficient will be inserted into the priority queue with his initial error (e_i). At each step of the algorithm the coefficient with the (updated) smallest error contribution will be removed from the queue and thresholded. All error contributions of the coefficients that are in the queue and are affected by the thresholding of c_i will be updated. Thus we need an implementation of the priority queue that supports fast extraction of the top element and fast location and update of an element in the queue. For our implementation we use the heap-based priority queue. It has the following properties:
1. Any node is greater than any value in the subtree.
2. It is a complete tree, except perhaps for the last row, height $O(\log N)$ - all operations take $O(\log N)$.
3. The heap can be implemented in array of size N without any need for additional storage.

The total initial calculation of error contributions takes $O(N\bar{U})$ time. The total time for thresholding coefficients with the smallest error contribution takes is $O(N \log N)$. Updating the approximate results of the queries takes on the average $O(N\bar{U})$ time. Updating the error contributions of the remaining coefficients takes $O(N\bar{U} \log N)$ time.

For every coefficient $c_{i'}$ whose error contribution is updated as a result of the thresholding of coefficient c_i, we need to update its position in the priority queue. Hence, the total number of priority queue operations is $\sum_{i=1}^{N}(1 + |U(c_i)| \log N) \log N \leq N \log N + w \log^3 N$, based on Lemma 1.

The theorem follows. ∎

4.2.1 The uniform workload case

The adaptive greedy algorithm has in general an advantage over the standard greedy algorithm even for the case in which the workload is taken from the uniform distribution. While the standard greedy algorithm is optimal for the MSE error, it is not optimal for the MRE and the adaptive greedy algorithm can provide better accuracy. Representing the uniform distribution using a workload of N^2 queries would be too expensive. the adaptive greedy algorithm can still build an effective synopsis for the MRE. When the training set is too small, the adaptive greedy algorithm may not be effective, because it could wrongfully assume some skew in the workload and optimize towards that skew. As demonstrated in the experimental results, there is a training set that is substantially smaller than N^2, but is still sufficiently large to obtain an effective utilization of the adaptive greedy algorithm.

4.3 The backward adaptive algorithm

We consider an alternative adaptive algorithm in which a candidate set is initially empty, and the algorithm iteratively selects a candidate for the inclusion based on the greatest improvement in the workload error (Figure 9). As in the previous algorithm, in the first step, the backward algorithm builds the workload matrix data structure from the given wavelet coefficients and the workload. The algorithm then calculates

```
BackwardAdaptiveThreshold(C, m, W)

Initialize
Build the workload matrix data structure
for each c_i ∈ C
  do calculate its (initial) error contribution (e_i)
Build a priority queue PQ for the coefficients
while |C| < m
  do { Extract the top coefficient (c_i) from the PQ
       Add c_i to C; C = C ∪ {c_i}
       for each q_j ∈ U(c_i)
         do Update approximate results (d̂_j)
       for each q_j ∈ U(c_i)
         do { for each c_i' ∈ D(q_j) ⋂ PQ
              do Update the error contributions (e_i') }
```

Figure 9: The backward adaptive thresholding algorithm

the initial error contribution for each coefficient in the workload matrix. In the next step, the algorithm builds the priority queue of the coefficients, based on the calculated initial error contribution. The main iteration loop of the algorithm is:

Step 1 Select the coefficient c_i with the *greatest* error contribution e_i from the priority queue.

Step 2 Add it to the synopsis C and update the approximate results d_j of all updated queries q_j in the workload matrix, that are affected by the inclusion of c_i to C ($q_j \in U(c_i)$).

Step 3 For each updated query q_j, update the error contributions e_i, of the query's coefficients c_i.

When the necessary number of coefficients is selected, the algorithm returns selected coefficients. The running time of this algorithm is better than the previous greedy algorithm, since $m \ll N$, yet its accuracy appears to be inferior, as observed in experimental studies.

The initial calculation of error contributions takes $O(N\bar{U})$ time as in the previous one. Restoring coefficients with the greatest error contribution takes a total of $O(m \log N)$ time, based on that each operation in the priority queue takes $\log N$ time. Updating the approximate results of the queries takes on the average $O(m\bar{U})$ time. Updating the error contributions of the remaining coefficients takes $O(m\bar{U} \log N)$ time. Thus the algorithm runs in $O((w + m + \frac{mw \log N}{N}) \log N)$ time with $O(w + N)$ space in addition to $O(N)$ of wavelet decomposition time.

4.4 A linear time algorithm

The overhead incurred in the adaptive algorithms due to the intensive priority queue operations can be effectively eliminated by relaxing the greedy algorithm, so that at every step the coefficient to be selected has the error contribution that is within a small ϵ approximation to the minimal coefficient at that stage. This can be implemented by using an approximated priority queue [23, 17], for which every operation takes constant, or near constant time (for very small ϵ).

The notion of approximation is as follows [23]: the operations are guaranteed to be consistent with the behavior of the corresponding exact data structure that operates on the elements after they are mapped by a fixed function f. For the multiplicatively approximate variant, the function f preserves the order of any two elements differing by at least a factor of some $1 + \epsilon$.

Let the elements be taken from a universe [1,U]. On an arithmetic RAM with b-words and under the standard assumptions that $b = \omega(\log U + \log n)$, where n is the input size, the running time required

```
ApproximateThreshold(C, m, W, ε)

Build the workload matrix data structure
for each c_i ∈ C
  do calculate its (initial) error contribution (e_i)
Build an approximate priority queue PQ for the coefficients
while |C| > m
  do { Extract the top coefficient (c_i) from PQ
       C = C \ {c_i}
       for each q_j ∈ U(c_i)
         do Update approximate results (d̂_j)
       for each q_j ∈ U(c_i)
         do { for each c_i' ∈ D(q_j) ⋂ PQ
              do Update the error contributions (e_i') }
```

Figure 10: The approximate adaptive thresholding algorithm

per operation in the approximate data structure is $O(1)$ for $\epsilon = 1/polylog(nU)$, and $O(\log\log n)$ for $\epsilon = 1/\exp polylog(n)$. The space requirements of the data structures are $O(\log(U)/\epsilon)$ and $O(U/\delta)$, respectively. The space can be reduced to be close to linear in the number of elements by using dynamic hashing.

The linear time thresholding algorithm reduces the approximate problem with all wavelet coefficients with universe size U to the exact problem with a smaller universe size $U' = 2\log_2(U)/\epsilon = O(\log_{1+\epsilon} U)$.

Each reduction is effectively reversible, yielding an equivalence between each approximate problem and the exact problem with a smaller universe. The equivalence holds generally for any numeric data type whose semantics depend only on the ordering of the elements (which is applicable to the thresholding process). Figure 10) presents the approximate adaptive thresholding algorithm. This algorithm is the same as the adaptive algorithm, described in Section 4.1, except that instead of the priority queue, we use approximate priority queue, such that each operation will take $O(1)$ time.

For $1/\epsilon < polylog(nU)$ and $w < N/\log^2 N$, we will have a linear-time algorithm, with $O(N + w\log^2 N)$ time and still with $O(N + w)$ space.

Using the approximate priority queue may slightly alter the order in which coefficients are selected for the thresholding. Compared to the adaptive thresholding algorithm, which will select in each step coefficient with the greatest error contribution, the approximate algorithm will select coefficient only within $1 + \epsilon$ of that error contribution. Note that the actual values of retained coefficients remain intact. The impact of this selection on consecutive selections is expected to be marginal.

As demonstrated in the experimental results, the use of approximate data structures is indeed indistinguishable in the accuracy of the synopses.

5 Self-tuning for the workload changes

An essential assumption behind the effectiveness of workload-based wavelet synopsis is that the training workload is a good one. That is, that future queries will be taken from a probability distribution that is similar to that of the training-workload. In reality, there may be some discrepancy between the training workload distribution and the actual distribution, either due to inherent sampling errors, or due to time-based changes in this distribution. It is therefore important that a workload-based synopsis would be robust to such discrepancies, and that it can adapt to changes in the workload skew. We will show experimental results regarding robustness in Section 7. In this section we present an algorithm that adapts the wavelet-synopses to updates in the query workload. Thus, it in fact adapts to the actual probability distribution in any given time.

5.1 A motivating example

We will continue the motivating example previously presented in Section 3.1. Figure 11 depicts a shift in the workload from the last 4 data values to the first 4 data values. It will introduce an increase in the MSE, if the synopsis remains unchanged. Figure 12 presents the same shift in the workload, but also an update to the coefficients retained in the synopses, which keeps the MSE as zero. This example demonstrates the importance of the dynamic updates in the synopsis, following changes in the original workloads.

5.2 The self-tuning algorithm

The self-tuning algorithm consists of two parts. The first is the initial thresholding that sets up the primary synopsis, as well as an auxiliary synopsis that will be used during the self-tuning phase. The second part is the self-tuning process in response to workload updates.

The input of the algorithm consists of: (1) an array $C = [c_1, ..., c_N]$ of the N Haar wavelet coefficients; (2) m – the number of coefficients to keep in the main histogram; (3) m' – the number of coefficients to keep in the auxiliary histogram; and (4) W - the training workload that consists of w queries. The output of the algorithm consists of an array $C' = [c_{i_1}, ..., c_{i_m}]$ of m non-zero retained coefficients in the main histogram and an array $C'' = [c_{i_{(m+1)}}, ..., c_{i_{(m+m')}}]$ of m' non-zero retained coefficients in the auxiliary histogram.

The self-tuning algorithm uses the error tree structure and the workload matrix presented before. Let T_0 be the initial error tree constructed from the initial data relation of an attribute at time t_0. The wavelet-based synopsis H_0 is a subset of internal nodes of T_0 obtained after applying the adaptive workload-based algorithm of Section 4 with the given initial workload W_0. Suppose that during a time interval from t_0 to t_1, the workload is updated with a new set W_1. A revised wavelet synopsis H_1 is computed by modifying H_0 based on W_1.

We use the following method of updating the workload: for w_1 new queries we will remove from W_0, of size w_0, $min(w_0, w_1)$ queries and replace them with the new ones. When a new workload is available, it will only affect error contributions e_i of the wavelet coefficients c_i that are used in the calculation of inserted and removed queries. In order to effectively compute such updated error contributions, we also keep the exact and approximate query results. This enables the recalculation of the new error contributions and the decision whether they have to be replaced by the previously thresholded coefficients.

The main 5 steps of the initialization part of the self-tuning algorithm are as follows:

Step 1 Let the initial error tree at time t_0 be T.

Step 2 Apply the adaptive thresholding algorithm and keep the $m + m'$ internal nodes of T.

Step 3 Select the m coefficient with the highest error contribution to be in synopsis H_0.

Step 4 Keep the other m' nodes and w_0 exact and approximate query results as an auxiliary histogram H' on disk.

Step 5 Set the *threshold value* to be the maximum error contribution among the m' nodes of H'.

The main 5 steps of the self-tuning algorithm in response to an updated workload are:

Step 1' Recalculate changes in the error contributions of H_0 wavelet coefficients.

Step 2' If this changes are less than our saved threshold value than exit.

Step 3' Recalculate the error contributions of the saved wavelet coefficients.

Step 4' Select the top m of them.

Step 5' Save the remaining coefficients on the disk with updated workload information.

We can extend this method also for two or more tiers of saved coefficients and keep threshold values for each tier. When the maximum error contribution of some tier becomes less than the thresholding value for the next tier, we move to the next tier and recalculate its error contributions.

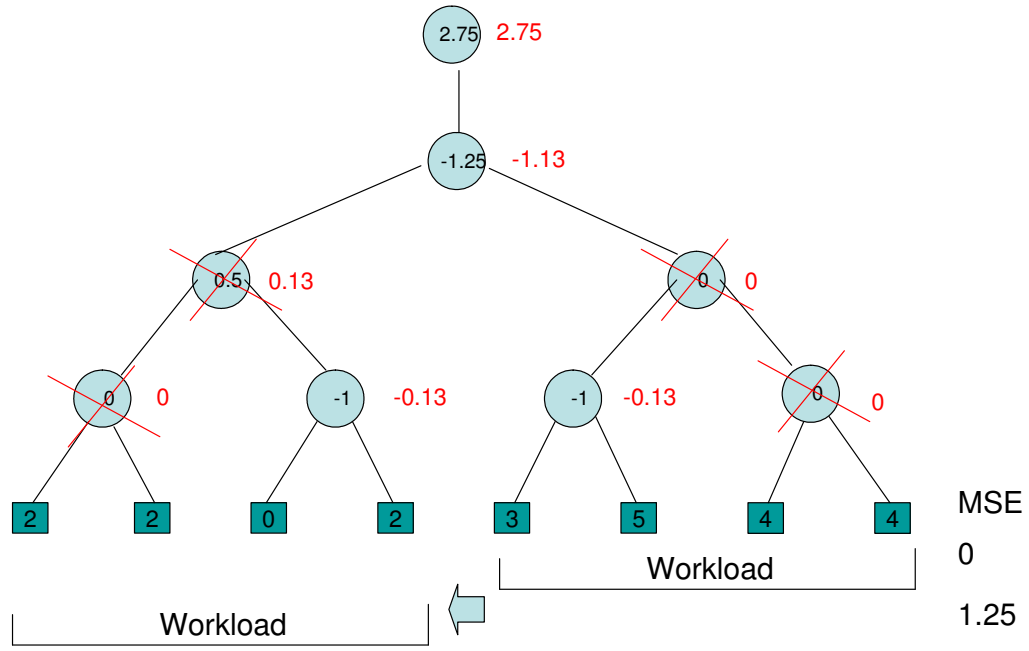

Figure 11: Self-tuning motivating example. Original synopsis.

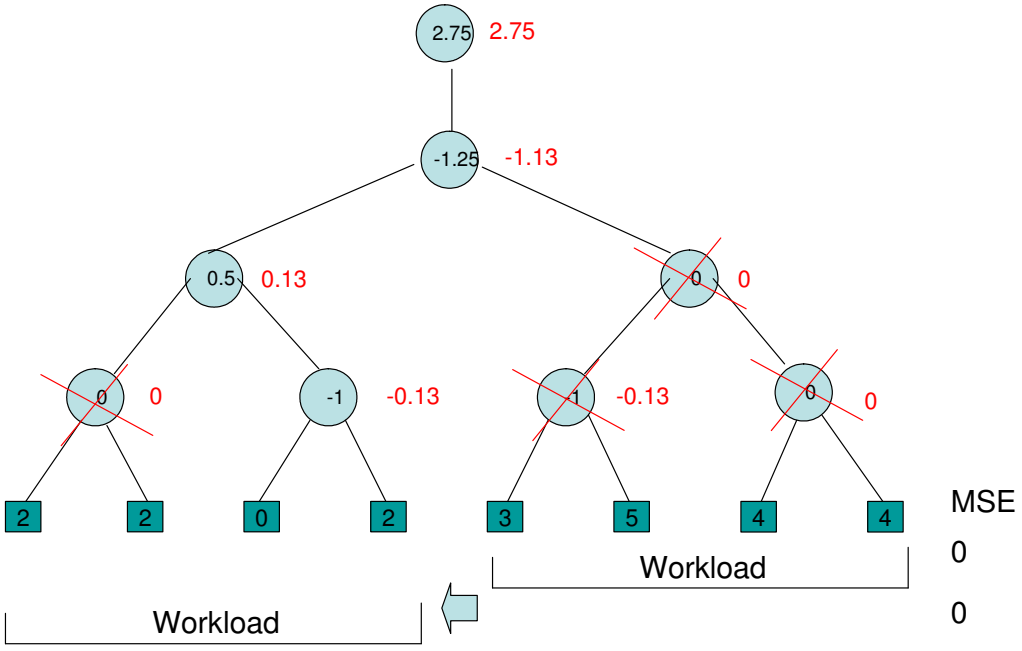

Figure 12: Self-tuning motivating example. Updated synopsis.

```
SelfTuningThreshold(C, m, W)

Build the workload matrix data structure
for each c_i ∈ C
  do calculate its (initial) error contribution (e_i)
Build a priority queue PQ for the coefficients
while |C| > m + m'
  do { Extract the top coefficient (c_i) from the PQ
       C = C \ {c_i}
       for each q_j ∈ U(c_i)
         do Update approximate results (d̂_j)
       for each q_j ∈ U(c_i)
         do { for each c_i' ∈ D(q_j) ⋂ PQ
              do Update the error contributions (e_i') }
Choose top m coefficients to be H_0
Save m' and w_o on the disk
Save the maximum error contribution of the m' coefficients, T
```

Figure 13: The self-tuning thresholding algorithm – initialization.

```
SelfTuningUpdate(C', C'', W')

for each c_i ∈ C' and C''
  do calculate its (initial) error contribution (e_i)
if all e_i < T
  do exit
Build a priority queue PQ for the coefficients
while |C| > m + m'
  do { Extract the top coefficient (c_i) from the PQ
       C = C \ {c_i}
       for each q_j ∈ U(c_i)
         do Update approximate results (d̂_j)
       for each q_j ∈ U(c_i)
         do { for each c_i' ∈ D(q_j) ⋂ PQ
              do Update the error contributions (e_i') }
Choose top m coefficients to be H_1
Save m' and w_1 on the disk
Save maximum error contribution of m' coefficients, T
```

Figure 14: The self-tuning part. C' is the main histogram, C'' is the auxiliary histogram, and W' is the workload update.

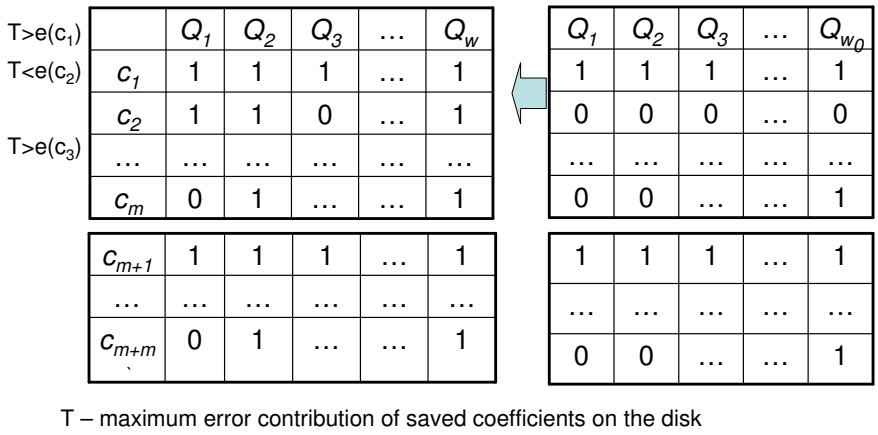

Figure 15: Self-tuning method

5.3 Complexity analysis

Recalculation of the error contributions can be done by $O((m + m') \log w)$ time. Selecting the top m coefficients takes $O(m \log(m + m'))$ time. The total execution time of this algorithm will therefore be $O((m + m') \log w + m \log(m + m'))$.

As demonstrated in the experimental results, the self-tuning algorithm can provide significant improvement over the (off-line) adaptive algorithm, when a change occurs in the distribution from which the queries are generated.

6 Multi-Dimensional wavelet synopses

Our basic workload-based adaptive algorithms can be easily extended to the multi-dimensional case. As in [21], we will transform the multi-dimensional data to the one-dimensional array. In this method, we first fix an ordering for the data dimensions and then apply the complete one-dimensional wavelet transform for each one-dimensional row of the data values along each original dimension. In such a case each multi-dimensional range query, can also be represented as a number of one dimensional queries. Thus the multi-dimensional case is essentially reduced to the one-dimensional case. The multi-dimensional workload-based algorithm is described in Figure 16.

All the previously presented modifications of the base algorithm, such as a backward or a linear-time approximation also apply to the multi-dimensional algorithm. The multi-dimensional algorithm can also be extended to support self-tuning for the changes in the workload.

The complexity of this algorithm is similar to the one-dimensional case, where N represent the total number of transformed coefficients and w represents the total number of transformed queries in the workload. The w can be potentially large and directly depends on the dimension order.

Experimental results demonstrate the significant improvement obtained for workload-based multidimensional wavelet synopses over previously considered multidimensional wavelet synopses.

```
MULTIDIMENSIONALTHRESHOLD(C, m, W)

Initialize
Build the workload matrix data structure
for each c_i ∈ C
  do calculate its (initial) error contribution (e_i)
Build a priority queue PQ for the coefficients
while |C| > m
  do { Extract the top coefficient (c_i) from the PQ
       C = C \ {c_i}
       for each q_j ∈ U(c_i)
         do Update approximate results (d̂_j)
       for each q_j ∈ U(c_i)
         do { for each c_i' ∈ D(q_j) ⋂ PQ
              do Update the error contributions (e_i') }
```

Figure 16: The multi-dimensional workload-based thresholding algorithm

7 Experimental study

In this section we describe the results from the experiments that we conducted using the presented algorithms. The objective is to verify the effectiveness of the algorithms and synopses, to quantify their scope, and to compare them to previous, non workload-based methods. For the experimental results, we have implemented and tested all the algorithms presented in the previous sections. For comparison purposes, we also implemented the basic wavelet algorithm from Section 3, that selects coefficients for the thresholding only by their level-based normalized value.

The experimental results can be summarized as follows:

1. **Different error metrics.** Figure 19 in Section 7.2 demonstrates significant improvements for the MSE, MRE and MAXRE errors with different synopses sizes over the basic algorithm for a real-life dataset. For instance, for a synopsis of size 20, this figure shows reduction in the MRE from 1.6 to 0.2 in favor of the adaptive algorithm. Figure 31 depicts similar improvements for a synthetic dataset; for a synopsis of size 500, the adaptive algorithm reduces the MSE from 4900 to 1200.

   Figure 20 in Section 7.2 demonstrates that with an increase of the synopses sizes from 10 to 100, the ratio between the MRE of the basic algorithm and the MRE of the adaptive algorithm increases from 10 to 30.

2. **Different workload types.** Figure 21 in Section 7.2 depicts that our adaptive algorithm outperforms the basic algorithm for different workload types. For example, for a synopsis of size 50, the adaptive algorithm achieves an MRE which is between 0.05 and 0.1 for the all workload types.

3. **Uniform workload.** Figure 22 shows that the adaptive nature of our algorithm allows it to outperform the basic one even for the uniform workloads for the MRE error and range-sum queries. For a synopsis of size 50, the adaptive algorithm has MRE= 0.05, where the basic algorithm has MRE= 0.85.

4. **Small training workloads.** We show that even small training workloads (100 queries) can help to achieve good error measures with a minor impact on the performance vs. non workload-based methods (Figure 23 in Section 7.2).

5. **Robustness.** Figure 24 shows that the algorithm is robust for deviations in the actual workload relative to the training workload. Only when the workload skew difference increases over 0.5, the MRE significantly increases.

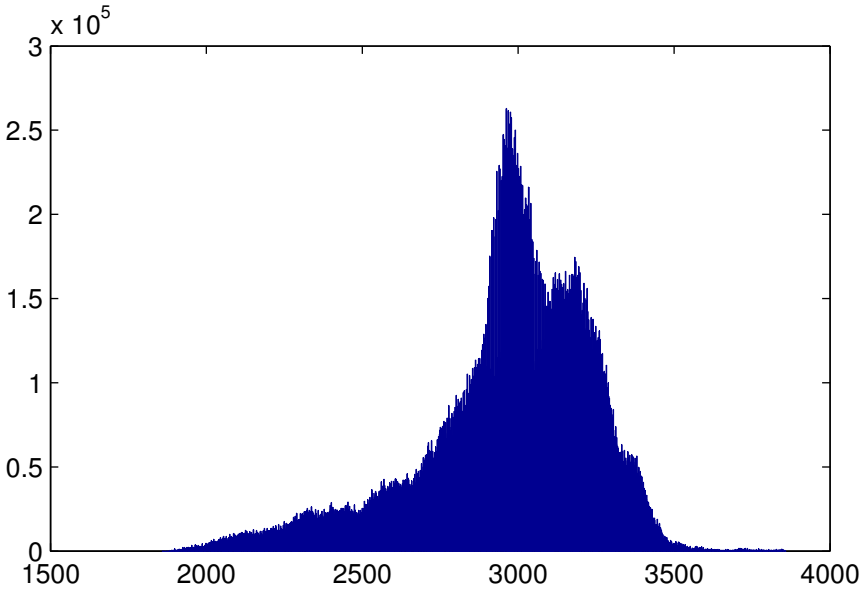

Figure 17: **CovType dataset distribution.** Filter attribute: "Elevation", data attribute: $Sum(\text{"}Aspect\text{"})$

6. **Linear time algorithms.** Figure 28 demonstrates that we can achieve a linear time algorithm with an insignificant impact on the approximation error.

7. **Self-tuning for updates in workload information.** Figure 29 shows about 100% improvement in error of the self-tuning algorithm when the query workload changes.

8. **Multi-dimensional data.** Figure 30 shows significant improvement over the basic multi-dimensional wavelet based algorithm.

7.1 Testbed and methodology

Datasets. For our experimental studies we used the real-life dataset provided by KDD Data of the University of California (http://kdd.ics.uci.edu) and the synthetic TPC-H dataset.

The KDD dataset is an actual forest cover type for 30x30 meter cells obtained from the US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from the US Geological Survey (USGS) and the USFS data. The real-life data set is a sum of the "Aspect" attribute of the Forest CoverType data set (581012 tuples, 54 attributes, 75.2MB data size). For the filter attribute, we used "Elevation" attribute, with 2048 distinct values (Figure 17).

The TPC-H dataset is a synthetic dataset, commonly used in the database performance analysis. For the synthetic dataset we used "ORDERS" table, and for the filter attribute we used the "O_CUSTKEY" attribute with 150000 distinct values, and for data attribute we used the order count for each customer (Figure 18).

Those datasets were also used for experimental results in previous methods and can provide a good reference point for comparison.

The framework. All experimental results in this paper are obtained from the τ-Synopses system [16]. τ-Synopses is a system designed to provide a run-time environment for remote execution of various synopses. It enables easy registration of new synopses from remote platforms, after which the system can manage these synopses, including triggering their construction, rebuilding and updating, and invoking them for approximate query processing. The system captures and analyzes query workloads, enabling its registered synopses to significantly boost their effectiveness (efficiency, accuracy, confidence), by exploiting workload information for synopses construction and update.

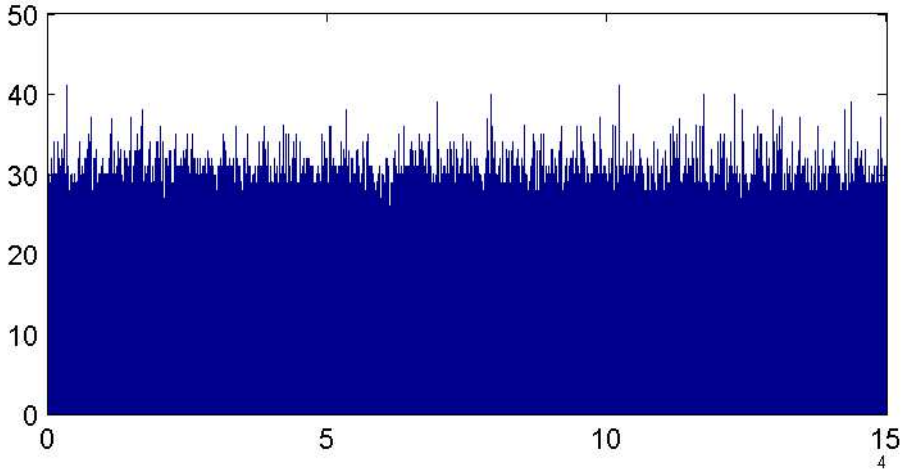

Figure 18: **TPC-H dataset distribution.** Filter attribute: "O_CUSTKEY", data attribute: $Sum(1)$

The workload. Queries of training and test workload sets were generated *independently* from the same distributions. As in previous methods, we define 4 different types of workloads: narrow uniform, wide uniform, narrow skewed and wide skewed workloads. The uniform workload means that all data is queried with the same frequency and the skewed workload means that some data is accessed more than other.

We used the following generated workloads in our experiments: each training workload with 100 queries and test workload with 500 queries for KDD and 200 and 1000 accordingly for TPC-H, from the following format:

```
SELECT Sum(Data) FROM Relation
WHERE filter > l AND filter < h .
```

For the uniform workloads, the lower limits are uniformly distributed over entire query ranges, and for the skewed workloads, lower limits have Zipfian distribution with $z = 0.5$. For the narrow workloads, query ranges are uniformly distributed in $[0, 50]$ and for the wide workloads the distribution is in $[0, 250]$ for KDD dataset and in $[0, 1000]$ for TPC-H.

Our wavelet synopses were constructed on the base relations and the query processing was performed completely over the thresholded error trees. Exact results were obtained by execution of the queries over the base relations in the original relational database (SQL Server 2000).

Error Metrics. As with previous work, the main metric we used for experimental results is the *mean relative error* (Eq. 4) during approximate query processing. For comparison with other methods we also used two additional error metrics: the *mean square error* (Eq. 3) and *maximum relative error* (Eq. 5).

7.2 Experimental results – real life data sets

Comparisons between the basic and adaptive algorithms. The purpose of these experiments is to compare the basic and the adaptive thresholding algorithms for different error metrics for the wide skewed workloads. The training and the test workloads were generated with the Zipfian distribution of 0.5 over whole range of the filter attribute [0, 2048]. The ranges of the queries were uniformly distributed between 0 and 250. For these workloads we built and evaluated the MSE, MRE and MAXRE errors for 10 different synopsis from each type with sizes from 10 to 100 with step 10.

Each plot in Figure 19 depicts different errors. The first plot shows that for the MSE the basic outperforms the adaptive algorithm only from the start and then the adaptive shows better performance. The second plot depicts that for the MRE the adaptive algorithm clearly wins the basic one. For the synopsis of size 20, it shows reduction in the MRE from 1.6 to 0.2 for the adaptive algorithm. The third plot shows the same behavior also for the MAXRE.

The adaptive algorithm demonstrates significant improvements for the MSE, MRE and MAXRE errors with different synopsis sizes over the basic algorithm for the real-life dataset.

Effects of different workload types. In these experiments we show how different workload types affect the MRE for the basic and the adaptive thresholding algorithms. We used 4 different types of training and test workloads: wide uniform, wide skewed, narrow uniform and narrow skewed. Each training workload was of size 100 and each test workload was of size 500. Synopses sizes were varied from 10 to 100.

The top plot in Figure 21 shows the MRE for different workload types. For example, for the synopsis of size 50, the adaptive algorithm achieves MRE between 0.05 and 0.1 for the all workload types. The bottom plot presents the ratio for the MRE between the basic and the adaptive methods. For the narrow skewed workload, the adaptive algorithm exhibits more than 100 times improvement in the MRE. We show in Figure 20 that even with increase of the synopsis size, the MRE ratio between the basic and the adaptive algorithms, increases from 10 to 30 times.

These figures depict that even for different workload types, our adaptive algorithm clearly outperforms the basic one.

Handling of the uniform workload case. The next experiments focus on how the basic and the adaptive methods handle uniform workloads. Figure 22 (top) shows how our adaptive algorithm handles uniform workload, compared to the basic algorithm, for the different training size according to the MSE. Because of the adaptive nature of the algorithm and its sufficiently large training size, our algorithm clearly outperforms the basic algorithm.

Figure 22 (bottom) shows how adaptive algorithm handles uniform workload, compared to the basic algorithm for the different training size according to the MRE. The adaptive algorithm explicitly optimizes synopses for the MRE as can be seen from the figure compared to the left figure even for the small training sizes.

Effects of the training set size. Figure 23 shows how the different sizes of the training workload affect the adaptive synopsis. For these experiments we used 4 different training wide skewed workloads of sizes 100, 200, 300 and 400. We show that even small training workloads (100 queries) can help to achieve good error measures for the MRE with a minor impact on performance vs. non workload-based methods

Robustness analysis. In these experiments we would like to experimentally verify the robustness of the adaptive workload-based wavelet synopses to errors in the workload prediction.

Figure 24 we show the robustness of our algorithms to changes in the actual workload with respect to the estimated workload. We measured our algorithms over a Zipfian distributed workload, and checked the approximation error as a function of the workload-estimation error. We show that the closest the Zipfian parameter to our estimation, the more accurate our answers; small errors in our estimation introduce only small errors in the quality of the approximation. Only when workload skew difference increases over 0.5, the MRE significantly increases.

Comparison with probabilistic synopses. Figure 25 shows how our algorithm behaves in comparison with the probabilistic wavelet synopses, MinRelVar, from [11]. As we did not have access to the original implementation of the MinRelVar algorithm, we used an implementation by Idan Spector and Iftach Ragoler, done as part of the class on Algorithms for Massive Data Sets (Spring 2003) at Tel Aviv University. As can be seen from the figure, the adaptive algorithm significantly more accurate than the probabilistic MinRelVar algorithm, especially for the small synopses sizes.

Comparison of execution times. Figure 26 shows how our algorithm behaves in comparison with the probabilistic wavelet synopses in terms of execution times. The running times of the adaptive algorithm is almost indistinguishable from the x-axis.

Backward workload-based algorithm. Figure 27 depicts experimental results for the backward adaptive workload-based algorithm, presented in Section 4.3. Recall that the backward greedy workload-based algorithm is much more run-time effective than the original workload-based algorithm. As can be seen from the figure, it outperforms the basic algorithm, but not in the same rate like the adaptive algorithm.

Linear time approximation algorithm. Figure 28 depicts experimental results for the linear time approximate workload-based algorithm. It shows that the impact of the approximation is insignificant. The

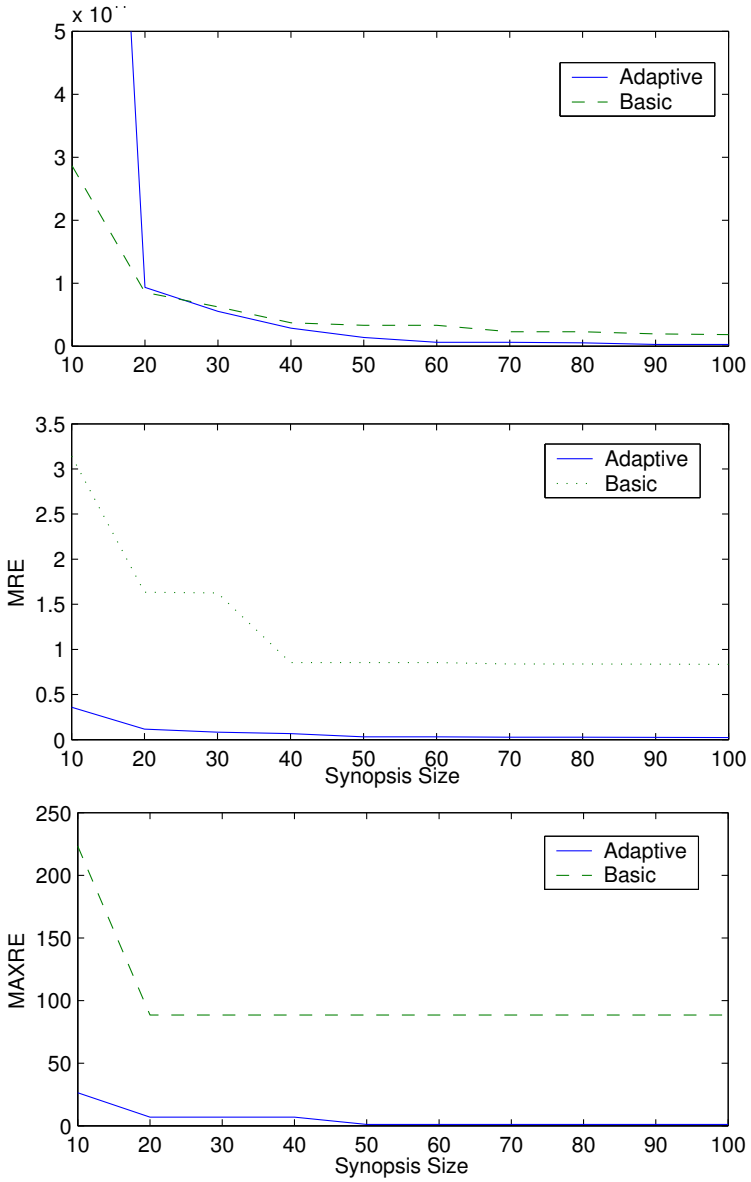

Figure 19: **Error for real-life data set.** Error of the adaptive vs. the basic algorithms, for synopses of sizes 10 to 100, for real-life data set. Top: MSE; middle: MRE; bottom: MAXRE.

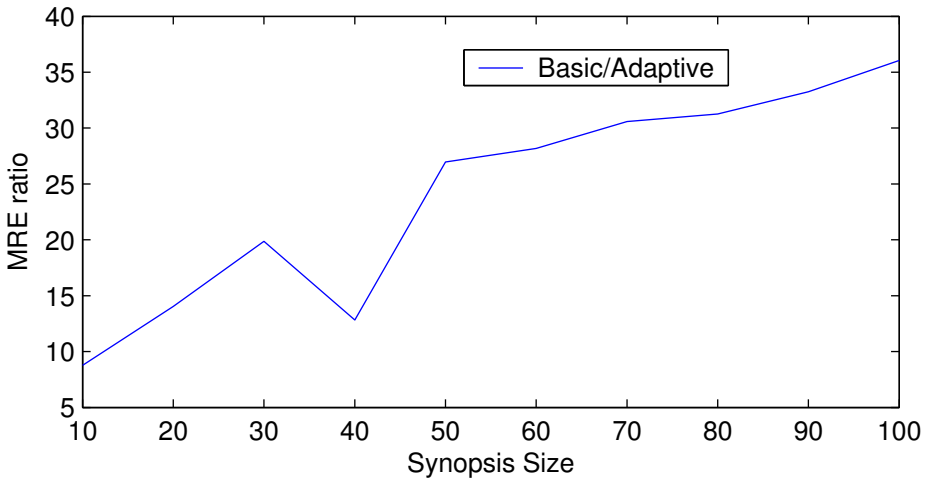

Figure 20: **MRE ratio as a function of synopsis size.** The ratio between the MRE of the basic algorithm and the MRE of the adaptive algorithm, for synopses of sizes between 10 to 100. In general the ratio increases with the synopsis size, reaching about 30.

bits number represents how much bits is reduced from the exact representation of the error contribution of coefficients in the approximate priority queue. The 0 bits represents exact algorithm.

Self-tuning greedy algorithm. A purpose of these experiments is to show the importance and effectiveness of the self-tuning algorithm for the changes in the workloads. The adaptive and the self-tuning synopses were built with the training workload with Zipfian skew 0.5 and then were tested with the test workload with skew 0.8. Figure 29 depicts experimental results for the self-tuning greedy workload-based algorithm, showing significant improvement over the off-line adaptive algorithm, when the probability distribution from which queries are generated is changing.

Our self-tuning algorithm shows a major improvement (about 100%) on the MRE.

Multi-dimensional adaptive algorithm. Figure 30 presents experimental results for the multi-dimensional adaptive workload-based algorithm, showing significant improvement over the basic multi-dimensional wavelet based algorithm. The latter was implemented by Oleg Davidov and Ran Dezent, as part of the class on Algorithms for Massive Data Sets (Spring 2003) at Tel Aviv University. The experiment executed over the two-dimensional dataset with *elevation* (1978 distinct values) and *slope* (53 distinct values) filter attributes and *aspect* as data attribute. In total there are 52452 distinct data values in the data set. The workload was created from wide skewed distribution with 500 queries.

7.3 Experimental results – synthetic data sets

Comparisons between the basic and adaptive algorithms. In Figure 31 we present comparison of the basic and the adaptive thresholding algorithms for different error metrics for the wide skewed workloads for the synthetic dataset. The training and the test workloads were generated with the Zipfian distribution of 0.5 over whole range of the filter attribute [0, 150000]. The ranges of the queries were uniformly distributed between 0 and 1000. For these workloads we built and evaluated the MSE, MRE and MAXRE for 10 different synopsis from each type with sizes from 500 to 5000.

Each plot in Figure 31 depicts different errors. The adaptive algorithm demonstrates significant improvements for the MSE, MRE and MAXRE metrics with different synopsis sizes over the basic algorithm for the synthetic dataset.

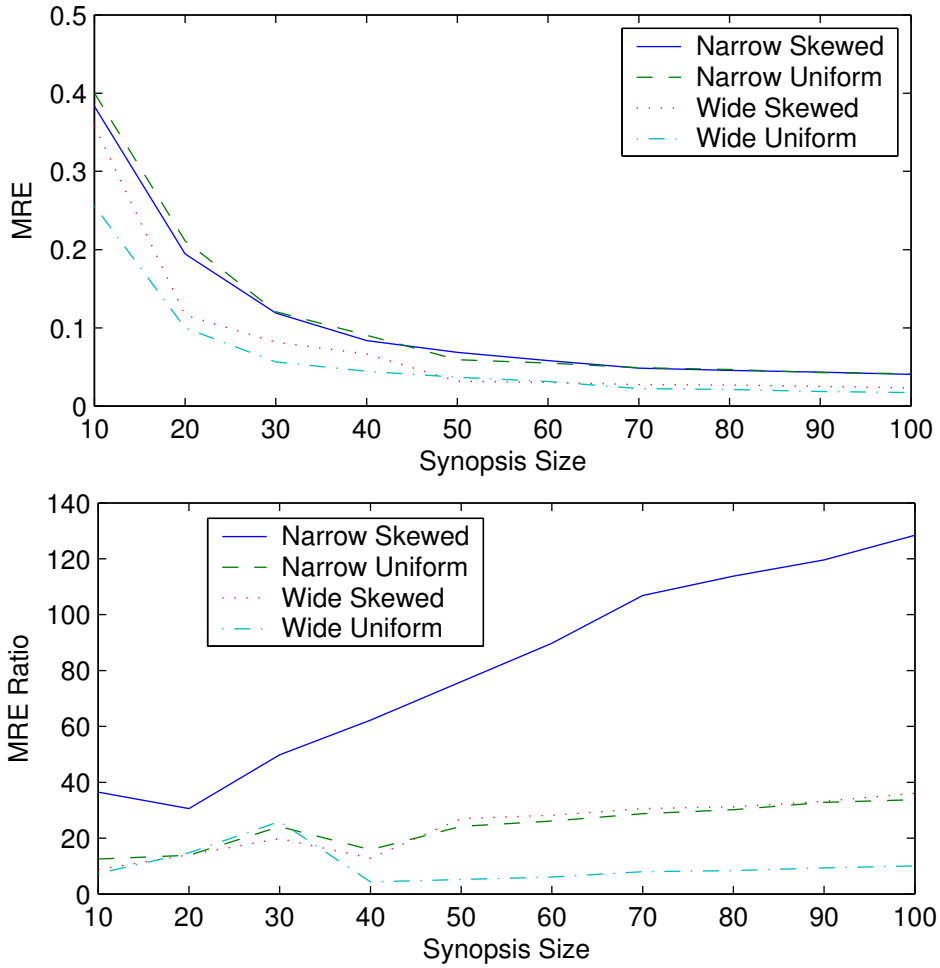

Figure 21: **MRE for various skews.** The top figure shows the MRE of the adaptive algorithm for different workload types. For example, for the synopsis of size 50, the adaptive algorithm achieves the MRE between 0.05 and 0.1 for the all workload types. The bottom figure presents the ratio for the MRE between the basic and the adaptive methods. For the narrow skewed workload, the adaptive algorithm exhibits between x30 and x130 improvement in the MRE.

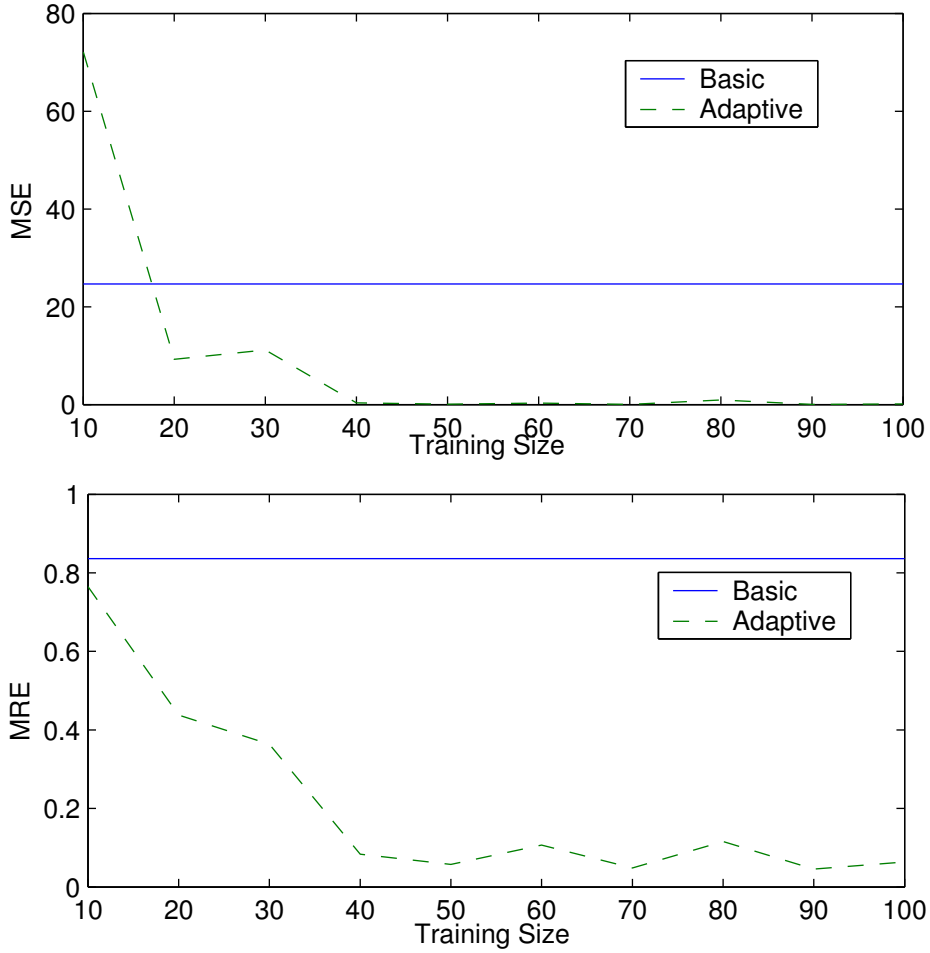

Figure 22: **Error as a function of workload training size.** The top figure describes effects of different training sizes for the uniform workload on the MSE. For small training sizes, the basic algorithm is better than the adaptive, because a small training set can erroneously indicate some skew in the workload. The bottom figure presents effects of different training sizes for the uniform workload on the MRE. The adaptive algorithm explicitly optimizes synopses for the MRE, as can be seen from the figure compared to the top figure, even for the small training sizes.

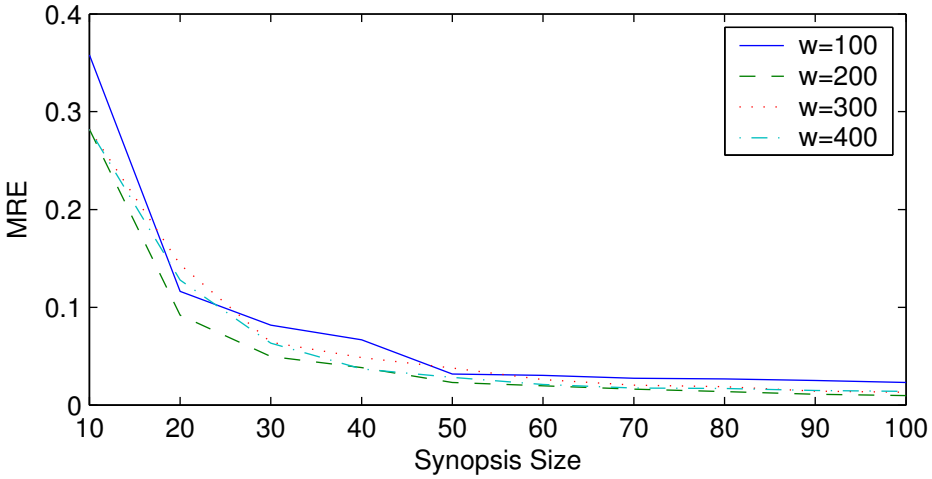

Figure 23: **Effect of the training size.** We can see that a training set of size 100 is enough for effective representation.

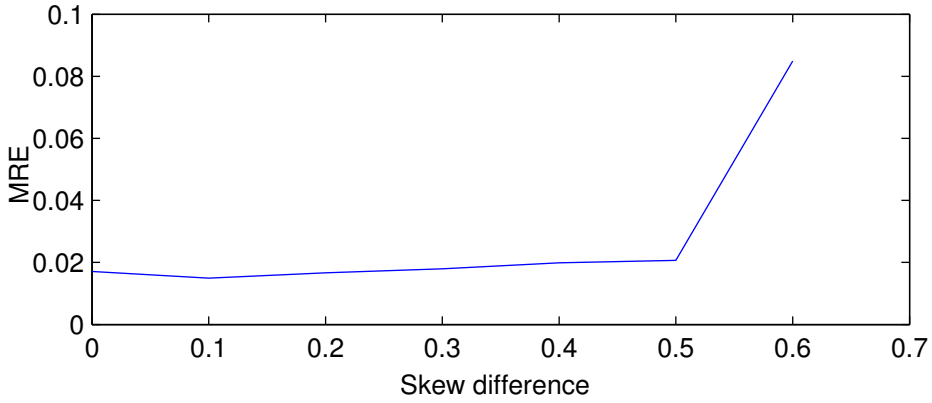

Figure 24: **Robustness to skew discrepancy between training and tested workloads.** Robustness of the adaptive algorithm to changes in the actual workload with respect to the estimated workload

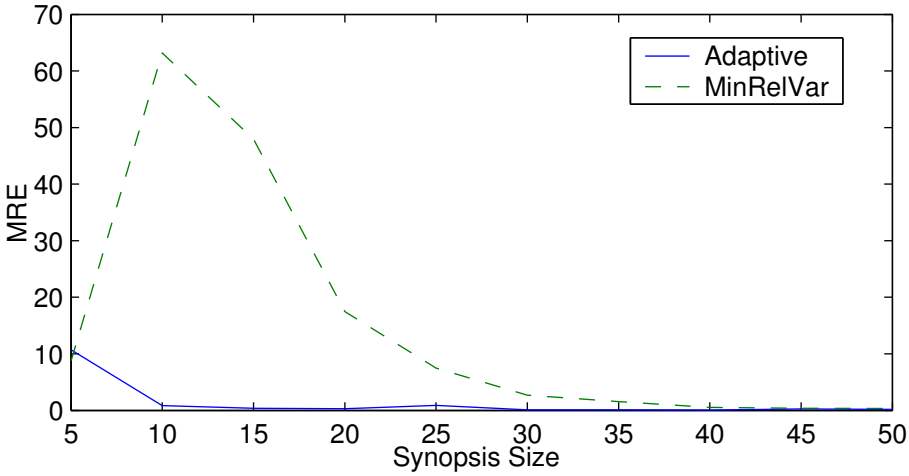

Figure 25: **The MRE of the workload-based synopsis vs. the probabilistic synopsis.** Comparison with the probabilistic MinRelVar wavelet synopses. As can be seen from the figure, the workload-based algorithm is significantly more accurate than the probabilistic MinRelVar algorithm, a specially for the small synopses sizes.

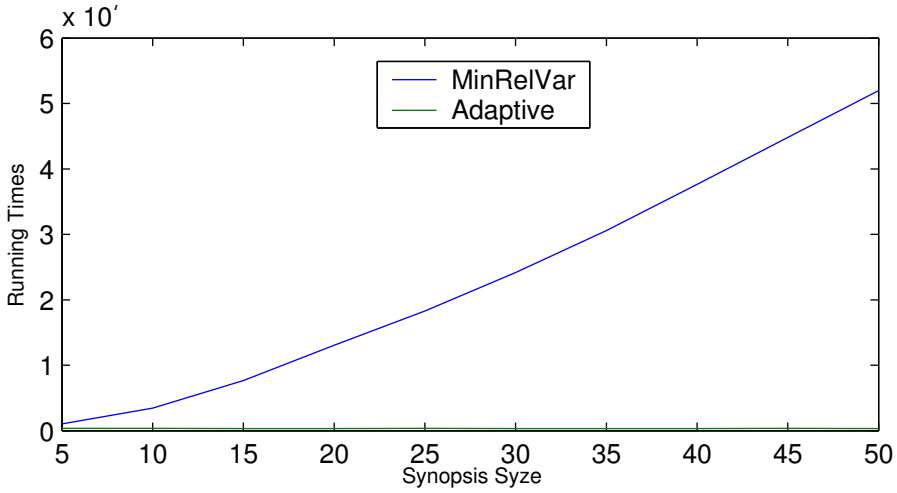

Figure 26: **The build time of the workload-based synopsis vs. the probabilistic synopsis.** Comparison of running times with the probabilistic wavelet synopses. The running times of the workload algorithm are almost indistinguishable from the x-axis.

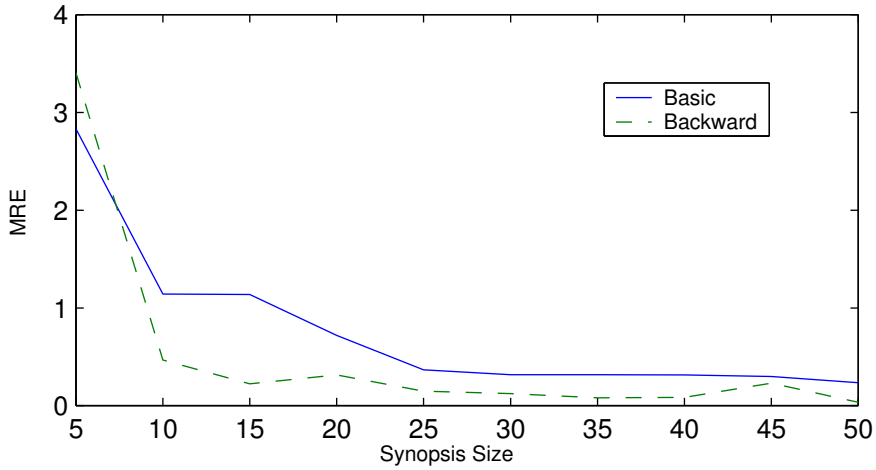

Figure 27: **The backward-adaptive based synopsis.** The backward workload-based greedy algorithm outperforms the basic algorithm (for example, for 15 retained coefficients it is 3 times better).

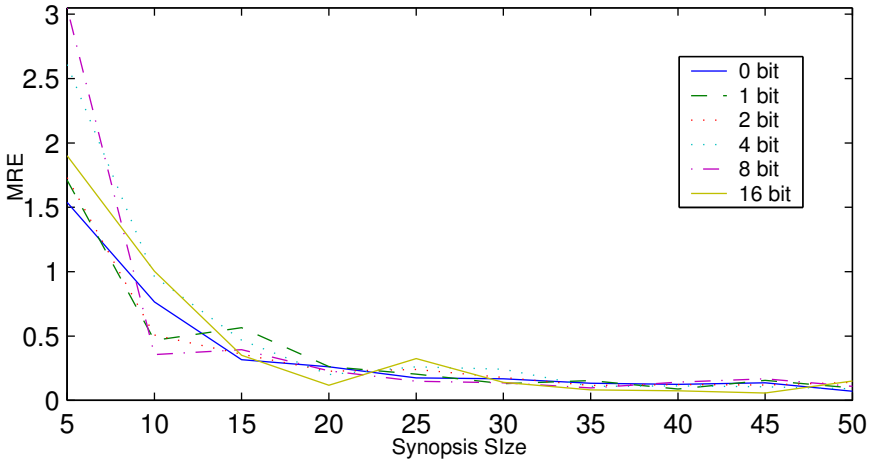

Figure 28: **Linear time approximation algorithm.** It shows that the impact of approximation is insignificant. The bits number represents the amount of bits which are reduced from the exact representation of the error contribution of coefficients in the approximate priority queue. The 0 bits represents the exact algorithm

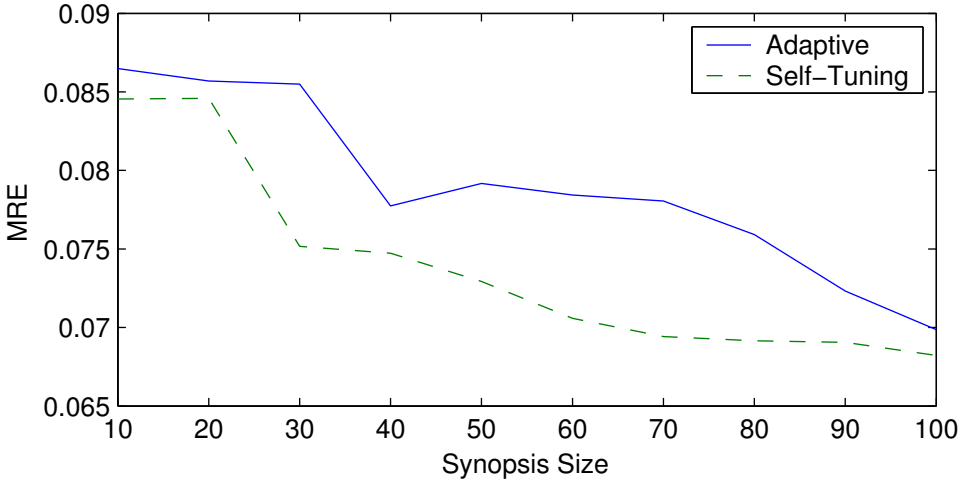

Figure 29: **Self-tuning vs. off-line.** Self-tuning vs. off-line adaptive algorithms for data with workload whose skew changes from 0.5 to 0.8 for synopses of sizes 10 to 100. The self-tuning algorithm has a significant advantage – about x2 improvement – in the MRE, over the off-line adaptive algorithm. the error metrics when the workload of the queries is changed

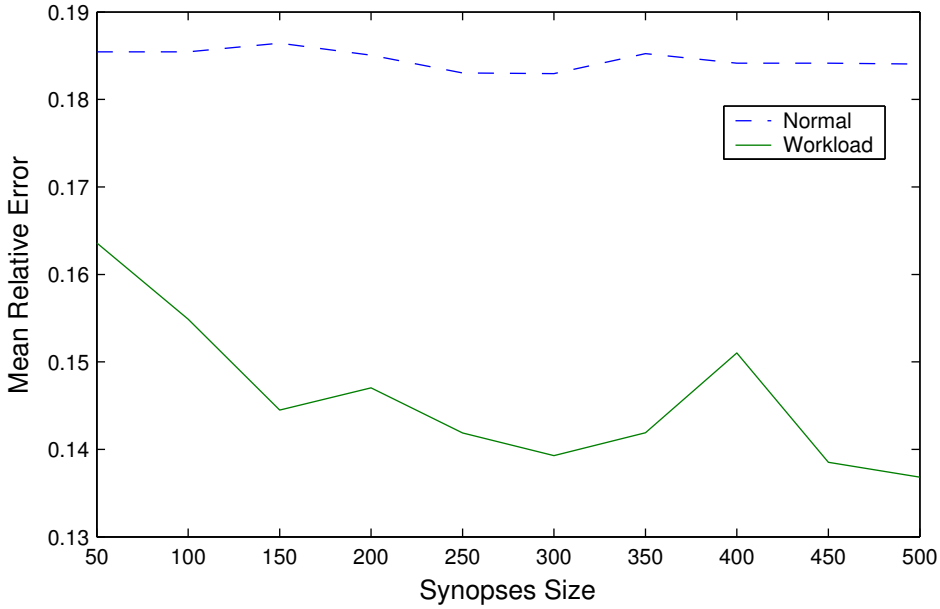

Figure 30: **Multi-dimensional wavelet synopses.** An adaptive workload-based vs. the basic greedy algorithm (MRE). The behavior of the multi-dimensional adaptive workload-based algorithm is more accurate than the basic algorithm.

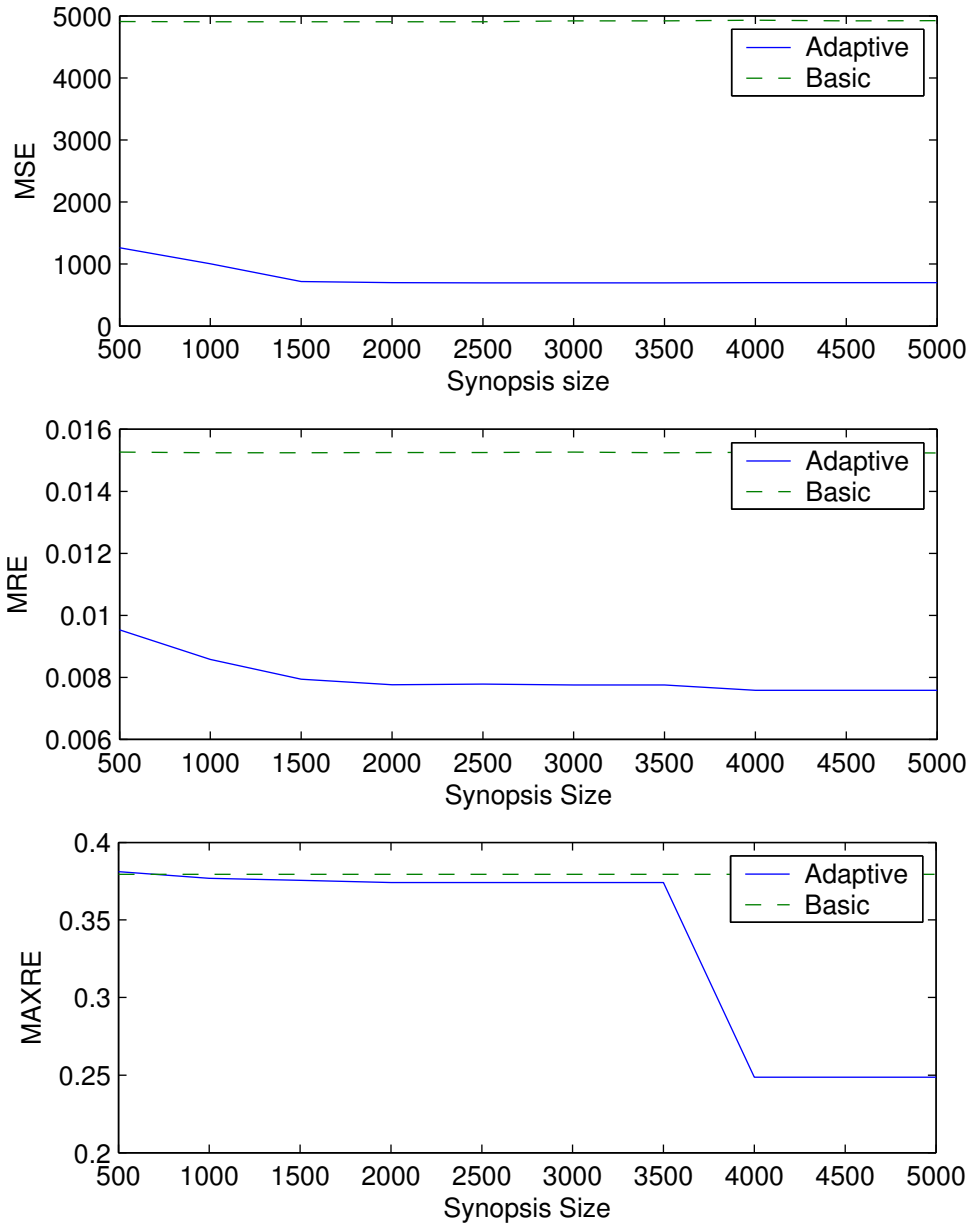

Figure 31: **Synthetic data sets.** The adaptive and basic algorithms for various synopses sizes with synthetic data. The adaptive algorithm demonstrates significant improvements for the MSE, MRE and MAXRE error metrics with different synopsis sizes over the basic algorithm for the synthetic dataset.

8 Conclusions

In this paper we proposed a method for building efficient and effective workload-based wavelet synopses using an adaptive workload-based algorithm. Our synopses can be adapted to any ad-hoc query workload and hence can give an improved performance for approximate query processing compared with previous, non-workload-based approaches.

We also showed a new method for self-tuning of workload-based wavelet synopses for updates in the workload. We extended our algorithms to the multi-dimensional case, obtaining workload-based multidimensional wavelet synopses We showed the effectiveness of our algorithms through extensive experimentation using synthetic and real-life data sets and a variety of synthetic query workloads.

Two natural extensions are to account for dynamic updates to the data sets [22], and employing probabilistic methods [11].

The algorithms proposed in this paper are with respect to a model that defines a *workload-based error-metric*. This model presents important theoretical and optimization questions regarding workload-based wavelet synopses.

Since the introduction of workload-based wavelet synopses in an earlier version of this work [15, 26] there has been some interesting progress in the research of regarding workload-based wavelet synopses [20, 24]. In [20] an efficient algorithm is presented for finding a weighted-haar basis and a synopsis that is optimal with respect to that basis, for a given workload of *point* queries. More recently, an optimal workload based wavelet synopses for the (non-weighted) haar basis was presented in [24], also with respect to a given workload of *point* queries. It leaves open the question of how to find an optimal synopsis for *range* queries, as well as questions about self-tuning synopses. Note that even though a range query can be answered using two point queries, synopses that are optimal for point queries could be far from optimal for range queries. Indeed, while the the basic greedy-based synopsis is known to be optimal for point queries in the (non-workload) MSE metric, it is not optimal for range queries. Similarly, while [12] presented a synopsis that is optimal in the MAXRE metric for point queries, it was shown in [18] that synopsis could be significantly less accurate than the synopses built using our adaptive algorithm in the MAXRE metric, when tested for range queries. Thus, the main open problem remains to find efficient algorithms that construct optimal workload-based wavelet synopses for a given workload or range queries.

References

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: building histograms without looking at data. In *Proceedings of the 1999 ACM SIGMOD*, pages 181–192, 1999.

[2] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *Proceedings of the 2001 ACM SIGMOD*, pages 275–286, 1999.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999. Special issue of selected papers from STOC'96.

[4] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 539–550. ACM Press, 2003.

[5] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: a multidimensional workload-aware histogram. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):211–222, 2001.

[6] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *VLDB Journal. Very Large Data Bases*, 10(2–3):199–223, 2001.

[7] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *ICDE*, pages 534–542, 2001.

[8] S. Chaudhuri, G. Das, and V. Narasayya. A robust, optimization-based approach for approximate answering of aggregate queries. In *Proceedings of the 2001 ACM SIGMOD*, 2001.

[9] A. Deligiannakis and N. Roussopoulos. Extended wavelets for multiple measures. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 229–240. ACM Press, 2003.

[10] V. Ganti, M.-L. Lee, and R. Ramakrishnan. ICICLES: Self-tuning samples for approximate query answering. In *The VLDB Journal*, pages 176–187, 2000.

[11] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *Proceedings of the 2002 ACM SIGMOD*, pages 476–487, 2002.

[12] M. Garofalakis and A. Kumar. Deterministic wavelet thresholding for maximum-error metrics. In *Proceedings of the 2004 ACM PODS international conference on on Management of data*, pages 166–176, 2004.

[13] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the 1998 ACM SIGMOD*, pages 331–342, 1998.

[14] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *External Memory Algorithms, DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization*, 50, 1999. American Mathematical Society.

[15] Y. Matias and L. Portman. Workload-based wavelet synopses. Technical report, Department of Computer Science, Tel Aviv University, 2003.

[16] Y. Matias and L. Portman. τ-Synopses: A system for run-time management of remote synopses (demo). In *Proceedings of the 2004 EDBT and ICDE*, 2004.

[17] Y. Matias, S. Sahinalp, and N. Young. Performance evaluation of approximate priority queues. In *DIMACS Fifth Implementation Challenge: Priority Queues, Dictionaries, and Point Sets*, October 1996.

[18] Y. Matias and D. Urieli. Improved implementation and experimental evaluation of the max-error optimized wavelet synopses. Technical report, School of Computer Science, Tel-Aviv University, 2004.

[19] Y. Matias and D. Urieli. On the optimality of the greedy heuristic in wavelet synopses for range queries. Technical report, School of Computer Science, Tel-Aviv University, 2005.

[20] Y. Matias and D. Urieli. Optimal workload-based weighted wavelet synopses. Technical report, School of Computer Science, Tel-Aviv University, 2005. Preliminary version in ICDT'05.

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

[22] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *The VLDB Journal*, pages 101–110, 2000.

[23] Y. Matias, J. S. Vitter, and N. E. Young. Approximate data structures with applications. In *Proc. 5th ACM-SIAM Symp. Discrete Algorithms*, pages 187–194, 1994.

[24] M. Muthukrishnan. Subquadratic algorithms for workload-aware haar wavelet synopses. In *Proc. of FSTTCS*, 2005.

[25] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the 1996 ACM SIGMOD*, pages 294–305, 1996.

[26] L. Portman. Workload-based wavelet synopses. Master's thesis, School of Computer Science, Tel Aviv University, 2003.

[27] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. Wavelets for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.

[28] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the 1999 ACM SIGMOD*, pages 193–204, 1999.

[29] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In G. Gardarin, J. C. French, N. Pissinou, K. Makki, and L. Bouganim, editors, *Proceedings of the 7th ACM International Conferences on Information and Knowledge Management*, pages 96–104, New York, U.S.A., 1998. Association for Computer Machinery.

[30] M. Wang. *Approximation and Learning Techniques in Database Systems.* Ph. D. dissertation, Duke University, 1999.