Approximate Query Processing Using Wavelets

Kaushik Chakrabarti[*]
University of Illinois
kaushikc@cs.uiuc.edu

Minos Garofalakis
Bell Laboratories
minos@bell-labs.com

Rajeev Rastogi
Bell Laboratories
rastogi@bell-labs.com

Kyuseok Shim
KAIST[†] and AITrc[‡]
shim@cs.kaist.ac.kr

Abstract

Approximate query processing has emerged as a cost-effective approach for dealing with the huge data volumes and stringent response-time requirements of today's decision-support systems. Most work in this area, however, has so far been limited in its applicability and query processing scope. In this paper, we propose the use of multi-dimensional wavelets as an effective tool for general-purpose approximate query processing in modern, high-dimensional applications. Our approach is based on building *wavelet-coefficient synopses* of the data and using these synopses to provide approximate answers to queries. We develop novel query processing algorithms that operate directly on the wavelet-coefficient synopses of relational tables, allowing us to process arbitrarily complex queries *entirely* in the wavelet-coefficient domain. This, of course, guarantees extremely fast response times since our approximate query execution engine can do the bulk of its processing over compact sets of wavelet coefficients, essentially postponing the expansion into relational tuples until the end-result of the query. An extensive experimental study with synthetic as well as real-life data sets establishes the effectiveness of our wavelet-based approach compared to sampling and histograms.

1. Introduction

Approximate query processing has recently emerged as a viable solution for dealing with the huge amounts of data, the high query complexities, and the increasingly stringent response-time requirements that characterize today's Decision-Support Systems (DSS) applications. Typically, DSS users pose very complex queries to the underlying Database Management System (DBMS) that require complex operations over Gigabytes or Terabytes of disk-resident data and, thus, take a very long time to execute to completion and produce exact answers. Due to the *exploratory nature* of many DSS applications, there are a number of scenarios in which an exact answer may not be required, and a user may in fact prefer a fast, approximate answer. For example, during a drill-down query sequence in ad-hoc data mining, initial queries in the sequence frequently have the sole purpose of determining the truly interesting queries and regions of the database [5]. Providing (reasonably accurate) approximate answers to these initial queries gives users the ability to focus their explorations quickly and effectively, without consuming inordinate amounts of valuable system resources.

Prior Work.[1] The strong incentive for approximate answers has spurred a flurry of research activity on approximate query processing techniques in recent years [1, 5, 6, 11, 17, 18]. The majority of the proposed techniques, however, have been somewhat limited in their *query processing scope*, typically focusing on specific forms of aggregate queries. Besides the range of queries, another crucial aspect of an approximate query processing technique is the employed *data reduction mechanism*; that is, the method used to obtain *synopses* of the data on which the approximate query execution engine can then operate. The methods explored in this context include *sampling* and, more recently, *histograms* and *wavelets*.

• *Sampling-based techniques* are based on the use of random samples as synopses for large data sets. Random samples of a data collection typically provide accurate estimates for aggregate quantities (e.g., `counts` or `averages`), as witnessed by the long history of successful applications of random sampling in population surveys [3] and selectivity estimation [8]. Sampling, however, suffers from two inherent limitations that restrict its applicability as an approximate query processing tool. First, a `join` operator applied on two uniform random samples results in a *non-uniform* sample of the join result that typically contains *very few tuples*, even when the join selectivity is fairly high [1]. Thus, `join` operations typically lead to significant degradations in the quality of an approximate aggregate. ("Join synopses" [1] provide a solution, but only for *foreign-key joins that are known beforehand*; that is, they cannot support arbitrary join queries over any schema.) Second, for a *non-aggregate* query, execution over random samples of the data is guaranteed to always produce a small subset of the exact answer which is often empty when `join`s are involved [1, 6].

• *Histogram-based techniques* have been studied extensively in the context of query selectivity estimation [12, 13] and, more recently, as a tool for providing approximate query answers [6, 11]. The very recent work of Ioannidis and Poosala [6] is the first to address the issue of ob-

[*] Work done while visiting Bell Laboratories. Author's current address: Univ. of California, Irvine, CA 92697.

[†] Korea Advanced Institute of Science and Technology.

[‡] Advanced Information Technology Research Center at KAIST.

**Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.**

[1] Due to space constraints, we omit a detailed discussion of related work; it can be found in the full version of this paper [2].

taining practical approximations to *non-aggregate* query answers, demonstrating how standard relational operators (like `join` and `select`) can be processed directly over histogram synopses of the data. The experimental results given in [6] prove that certain classes of histograms can provide higher-quality approximate answers compared to random sampling, when considering simple queries over low-dimensional data (one or two dimensions). It is a well-known fact, however, that histogram-based approaches become problematic when dealing with the high-dimensional data sets that are typical of modern DSS applications. The reason is that, as the dimensionality of the data increases, both the *storage overhead* (i.e., number of buckets) and the *construction cost* of histograms that can achieve reasonable error rates increase in an explosive manner [17]. This problem is further exacerbated by `join` operations that can cause the dimensionality of intermediate query results (and the corresponding histograms) to explode.

• *Wavelet-based techniques* provide a mathematical tool for the hierarchical decomposition of functions, with a long history of successful applications in signal and image processing [7, 10, 16]. Recent studies have also demonstrated the applicability of wavelets to selectivity estimation [9] and the approximation of range-sum queries over OLAP data cubes [17, 18]. Briefly, the idea is to apply wavelet decomposition to the input data collection (attribute column(s) or OLAP cube) to obtain a compact data synopsis that comprises a select small collection of *wavelet coefficients*. The results of Vitter et al. [17, 18] have clearly shown that wavelets can be very effective in handling aggregates over high-dimensional OLAP cubes, while avoiding the high construction costs and storage overheads of histograming techniques. Nevertheless, the focus of these earlier studies has always been on a very specific form of queries (i.e., range-sums) over a single OLAP table. Thus, the problem of whether wavelets can provide a solid foundation for general-purpose approximate query processing has hitherto been left unanswered.

Our Contributions. In this paper, we significantly extend the scope of earlier work on approximate query answers, establishing the viability and effectiveness of wavelets as a generic approximate query processing tool for modern, high-dimensional DSS applications. More specifically, we propose a novel approach to general-purpose approximate query processing that consists of two basic steps. First, multi-dimensional Haar wavelets are employed to efficiently obtain effective, compact synopses of general relational tables. Second, using novel query processing algorithms, standard (aggregate and non-aggregate) SQL operators are applied *directly* over the wavelet-coefficient synopses of the data to obtain fast and accurate approximate query answers. The crucial observation here is that, as we demonstrate in this work, our approximate query execution engine can do all of its processing *entirely in the wavelet-coefficient domain*; that is, both the input(s) and the output of our query processing operators are compact collections of wavelet coefficients capturing the underlying relational data. This, of course, implies that we can delay the expansion of our wavelet-coefficient synopses to the very end of an arbitrarily complex query, thus allowing for extremely fast approximate query processing. The contributions of our work are summarized as follows.

• **New, I/O-Efficient Wavelet Decomposition Algorithm for Relational Tables.** The methodology developed in this paper is based on a different form of the multi-dimensional Haar transform than that employed by Vitter et al. [17, 18]. As a consequence, the decomposition algorithms proposed by Vitter and Wang [17] are not applicable. We address this problem by developing a novel, I/O-efficient algorithm for building the wavelet-coefficient synopsis of a relational table. The worst-case I/O complexity of our algorithm matches that of the best algorithms of Vitter and Wang, requiring only a logarithmically small number of passes over the data. Furthermore, there exist scenarios (e.g., when the table is stored in *chunks* [4, 15]) under which our decomposition algorithm can work in a *single pass* over the input.

• **Novel Query Processing Algebra for Wavelet-Coefficient Data Synopses.** We propose a new algebra for approximate query processing that operates *directly over the wavelet-coefficient synopses of relations*, while guaranteeing the correct relational operator semantics. Our algebra operators include the conventional aggregate and non-aggregate SQL operators, like `select`, `join`, `sum`, and `average`. Based on the semantics of Haar wavelet coefficients, we develop novel query processing algorithms for these operators that work *entirely* in the wavelet-coefficient domain. This allows for extremely fast response times, since our approximate query execution engine can do the bulk of its processing over compact wavelet-coefficient synopses, essentially postponing the expansion into relational tuples until the end-result of the query.

• **Extensive Experiments Validating our Approach.** We have conducted an extensive experimental study with synthetic as well as real-life data sets to determine the effectiveness of our wavelet-based approach compared to sampling and histograms. Our results demonstrate that (1) the quality of approximate answers obtained from our wavelet-based query processor is, in general, better than that obtained by either sampling or histograms for a wide range of `select`, `project`, `join`, and aggregate queries, (2) query execution-time speedups of more than two orders of magnitude are made possible by our approximate query processing algorithms; and (3) our wavelet decomposition algorithm is extremely fast and scales linearly with the size of the data.

2. Building Synopses of Relational Data Using Multi-Dimensional Wavelets

2.1 Background: The Wavelet Decomposition

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 together with detail coefficients that influence the function at various scales [16]. The wavelet decomposition has excellent energy compaction and de-correlation properties, which can be used to effectively generate compact representations that exploit the structure of data.

The work in this paper is based on the multi-dimensional *Haar wavelet* decomposition. Haar wavelets are conceptually simple, very fast to compute, and have been found to perform well in practice for a variety of applications ranging from image editing and querying [10, 16] to selectivity estimation and OLAP approximations [9, 17]. In this section, we discuss Haar wavelets in both one and multiple dimensions.

One-Dimensional Haar Wavelets. Suppose we are given a one-dimensional data vector A containing the following four values $A = [2, 2, 5, 7]$. The Haar wavelet transform of A can be computed as follows. We first average the values together pairwise to get a new "lower-resolution" representation of the data with the following average values $[2, 6]$. In other words, the average of the first two values (that is, 2 and 2) is 2 and that of the next two values (that is, 5 and 7) is 6. Obviously, some information has been lost in this averaging process. To be able to restore the original four values of the data array, we need to store some *detail coefficients*, that capture the missing information. In Haar wavelets, these detail coefficients are simply the differences of the (second of the) averaged values from the computed pairwise average. Thus, in our simple example, for the first pair of averaged values, the detail coefficient is 0 since 2-2 =0, while for the second we need to store -1 since $6 - 7 = -1$. Note that it is possible to reconstruct the four values of the original data array from the lower-resolution array containing the two averages and the two detail coefficients. Recursively applying the above pairwise averaging and differencing process on the lower-resolution array containing the averages, we get the following full decomposition.

Resolution	Averages	Detail Coefficients
2	[2, 2, 5, 7]	–
1	[2, 6]	[0, -1]
0	[4]	[-2]

We define the *wavelet transform* (also known as the *wavelet decomposition*) of A to be the single coefficient representing the overall average of the data values followed by the detail coefficients in the order of increasing resolution. Thus, the one-dimensional Haar wavelet transform of A is given by $W_A = [4, -2, 0, -1]$. Each entry in W_A is called a *wavelet coefficient*. The main advantage of using W_A instead of the original data vector A is that for vectors containing similar values most of the detail coefficients tend to have very small values. Thus, eliminating such small coefficients from the wavelet transform (i.e., treating them as zeros) introduces only small errors when reconstructing the original data, giving a very effective form of lossy data compression.

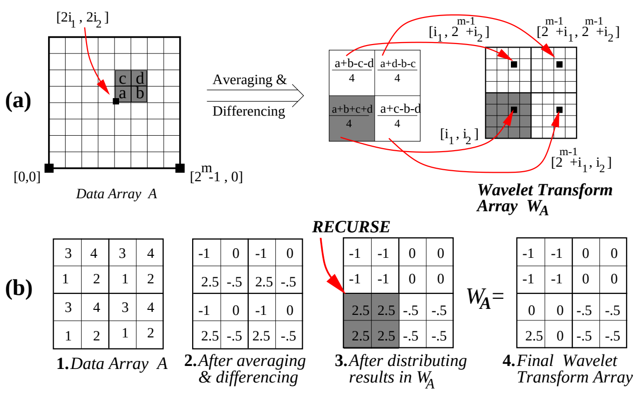
Note that, intuitively, wavelet coefficients carry different weights with respect to their importance in rebuilding the original data values. For example, the overall average is obviously more important than any detail coefficient since it affects the reconstruction of all entries in the data array. In order to equalize the importance of all wavelet coefficients, we need to *normalize* the final entries of W_A appropriately. This is achieved by dividing 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). Thus, the normalized wavelet transform for our example data array becomes $W_A = [4, -2, 0, -1/\sqrt{2}]$.

Multi-Dimensional Haar Wavelets. There are two common methods in which Haar wavelets can be extended to transform the data values in a *multi-dimensional* array. Each of these transformations is a generalization of the one-dimensional decomposition process described above. To simplify the exposition to the basic ideas of multi-dimensional wavelets, we assume all dimensions of the input array to be of equal size.

The first method is known as *standard decomposition*. In this method, we first fix an ordering for the data dimensions (say, $1, 2, \ldots, d$) and then proceed to apply the complete one-dimensional wavelet transform for each one-dimensional "row" of array cells along dimension k, for all $k = 1, \ldots, d$. The standard Haar decomposition forms the basis of the recent results of Vitter et al. on OLAP data cube approximations [17, 18].

The work presented in this paper is based on the second method of extending Haar wavelets to multiple dimensions, namely the *nonstandard decomposition*. Abstractly, the nonstandard Haar decomposition alternates between dimensions during successive steps of pairwise averaging and differencing: given an ordering for the data dimensions ($1, 2, \ldots, d$), we perform *one step of pairwise averaging and differencing* for each one-dimensional row of array cells along dimension k, for each $k = 1, \ldots, d$. (The results of earlier averaging and differencing steps are treated as data values for larger values of k.) This process is then repeated recursively only on the quadrant containing averages across all dimensions. One way of conceptualizing (and implementing [10]) this procedure is to think of a $2 \times 2 \times \cdots \times 2 (= 2^d)$ hyper-box being shifted across the data array, performing pairwise averaging and differencing, distributing the results to the appropriate locations of the wavelet transform array W_A (with the averages for each box going to the "lower-left" quadrant of W_A) and, finally, recursing the computation on the lower-left quadrant of W_A. This procedure is demonstrated pictorially for a $2^m \times 2^m$ data array A in Figure 1(a). More specifically, Figure 1(a) shows the pairwise averaging and differencing step for one positioning of the 2×2 box with its "root"(i.e., lower-left corner) located at the coordinates $[2i_1, 2i_2]$ of A followed by the distribution of the results in the wavelet transform array. This step is repeated for every possible combination of i_j's, $i_j \in \{0, \ldots, 2^{m-1} - 1\}$. Finally , the process is recursed only on the lower-left quadrant of W_A

(containing the averages collected from all boxes). A detailed description of the nonstandard Haar decomposition can be found in any standard reference on the subject (e.g., [7, 16]).



3	4	3	4
1	2	1	2
3	4	3	4
1	2	1	2

-1	0	-1	0
2.5	-.5	2.5	-.5
-1	0	-1	0
2.5	-.5	2.5	-.5

-1	-1	0	0
-1	-1	0	0
2.5	2.5	-.5	-.5
2.5	2.5	-.5	-.5

-1	-1	0	0
-1	-1	0	0
0	0	-.5	-.5
2.5	0	-.5	-.5

Figure 1. Non-standard decomposition in two dimensions. (a) Computing and distributing pairwise averages and differences. (b) Example decomposition of a 4×4 array.

Example 2.1: Consider the 4×4 array A shown in Figure 1(b.1). Figure 1(b.2) shows the result of the first horizontal and vertical pairwise averaging and differencing on the 2×2 hyper-boxes of the original array. The averages of the values for each positioning of the hyper-box are assigned to the 2×2 lower-left quadrant of the wavelet transform array W_A, while the detail coefficients are distributed in the three remaining 2×2 quadrants of W_A, as shown in Figure 1(b.3). The process is then recursively performed on the lower-left quadrant of W_A, resulting in detail coefficients of 0 and an average value of 2.5 which is stored in the lower-left entry of W_A. The final wavelet transform array W_A is depicted in Figure 1(b.4). ∎

As noted in the wavelet literature, both methods for extending one-dimensional Haar wavelets to higher dimensionalities have been used in a wide variety of application domains and, to the best of our knowledge, none has been shown to be uniformly superior. Our choice of the nonstandard method was mostly motivated by our earlier experience with nonstandard two-dimensional Haar wavelets in the context of effective image retrieval [10]. An advantage of using the nonstandard transform is that, as we explain later in the paper, it allows for an efficient representation of the sign information for wavelet coefficients. This efficient representation stems directly from the construction process for a nonstandard Haar basis [16]. (We often omit the "nonstandard" qualification in what follows.)

Multi-Dimensional Haar Coefficients: Semantics and Representation. Consider a wavelet coefficient W generated during the multi-dimensional Haar decomposition of a d-dimensional data array A. From a mathematical standpoint, this coefficient is essentially a multiplicative factor for an appropriate *Haar basis function* when the data in A is expressed using the d-dimensional Haar basis [16]. The d-dimensional Haar basis function corresponding to W is defined by (1) a *d-dimensional rectangular support region* in A that essentially captures the region of A's cells that W contributes to during reconstruction; and (2) the *quadrant sign information* that defines the sign ($+$ or $-$) of W's contribution (i.e., $+W$ or $-W$) to any cell contained in a given quadrant of its support rectangle. (Note that the wavelet decomposition process guarantees that this sign can only change across quadrants of the support region.) As an example, Figure 2(a) depicts the support regions and signs of the sixteen nonstandard, two-dimensional Haar basis functions for coefficients in the corresponding locations of a 4×4 wavelet transform array W_A. The blank areas for each coefficient correspond to regions of A whose reconstruction is independent of the coefficient, i.e., 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 Fig. 2(a)) only to values in A's upper right quadrant. Figure 2(a) 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 [16].

Example 2.2: In light of Figure 2(a), let us now revisit Example 2.1 and consider how the entries of W_A contribute to the reconstruction of values in A. As we have already observed, coefficient $W_A[0,0] = 2.5$ is the overall average that contributes $+2.5$ to the reconstruction of all sixteen data values in A. On the other hand, the detail coefficient $W_A[0,2] = -1$ affects the reconstruction of only the four values in the lower-left quadrant of A, contributing -1 to $A[0,0]$ and $A[1,0]$, and $-(-1) = +1$ to $A[0,1]$ and $A[1,1]$. Similarly, the detail coefficient $W_A[2,0] = -.5$ contributes $-.5$ to $A[0,0]$ and $A[0,1]$, and $+.5$ to $A[1,0]$ and $A[1,1]$. For example, based on Figure 2(a), the data value $A[0,1]$ can be reconstructed using the formula: $A[0,1] = +W_A[0,0] + W_A[0,1] + W_A[1,0] + W_A[1,1] - W_A[0,2] + W_A[2,0] - W_A[2,2] = 2.5 - (-1) + (-.5) = 3$. ∎

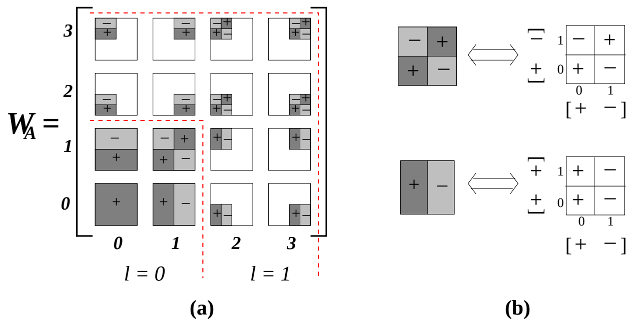


Figure 2. (a) Support regions and signs for the sixteen nonstandard two-dimensional Haar basis functions. (b) Representing quadrant sign information for coefficients using "per-dimension" sign vectors.

To simplify the discussion in this paper, we abstract away the distinction between a coefficient and its corresponding basis function by representing a Haar wavelet coefficient with the triple $W = \langle R, S, v\rangle$, where:

(1) $W.R$ is the *d-dimensional support hyper-rectangle of* W enclosing all the cells in the data array A to which W contributes (i.e., the support of the corresponding basis function). We represent this hyper-rectangle by its low and high boundary values (i.e., starting and ending array cells) along each dimension j, $1 \leq j \leq d$; these are denoted by $W.R.bound[j].lo$ and $W.R.bound[j].hi$, respectively. Thus, the coefficient W contributes to each data cell $A[i_1, \ldots, i_d]$ satisfying the condition $W.R.bound[j].lo \leq i_j \leq W.R.bound[j].hi$ for all dimensions j, $1 \leq j \leq d$. The space required to store the support hyper-rectangle of a coefficient is $2 \log N$ bits, where N denotes the total number of cells of A.

(2) $W.S$ stores the *sign information for all d-dimensional quadrants of* $W.R$. Storing the quadrant sign information directly would mean a space requirement of $O(2^d)$, i.e., proportional to the number of quadrants of a d-dimensional hyper-rectangle. Instead, we use a more space-efficient representation of the quadrant sign information (using only $2d$ bits) that exploits the regularity of the nonstandard Haar transform. The basic observation here is that a nonstandard d-dimensional Haar basis is formed by scaled and translated products of d one-dimensional Haar basis functions [16]. Thus, our idea is to store a 2-bit *sign vector* for each dimension j, that captures the sign variation of the corresponding one-dimensional basis function. The two elements of the sign vector of coefficient W along dimension j are denoted by $W.S.sign[j].lo$ and $W.S.sign[j].hi$, and contain the sign that corresponds to the lower and upper half of $W.R$'s extent along dimension j, respectively. Given the sign vectors along each dimension and treating a sign of $+$ $(-)$ as being equivalent to $+1$ (resp., -1), the sign for each d-dimensional quadrant can be computed as the product of the d sign-vector entries that map to that quadrant; that is, following exactly the basis construction process. (We will continue to make use of this "+1/-1" interpretation of signs throughout the paper.) Our sign-computation methodology is depicted in Figure 2(b) for two example coefficient hyper-rectangles from Figure 2(a).

(3) $W.v$ is the *(scalar) magnitude of coefficient W*. This is exactly the quantity that W contributes (either positively or negatively, depending on $W.S$) to all data array cells enclosed in $W.R$.

Thus, our view of a d-dimensional Haar wavelet coefficient is that of a d-dimensional hyper-rectangle with a magnitude and a sign that may change across quadrants. Note that, by the properties of the nonstandard Haar decomposition, given *any pair* of coefficients, their hyper-rectangles are either *completely disjoint* or one is *completely contained* in the other; that is, coefficient hyper-rectangles cannot *partially overlap*. It is precisely these containment properties coupled with our sign-vector representation of quadrant signs that enable us to efficiently perform `join` operations directly over wavelet-coefficient synopses.

2.2 Building Wavelet-Coefficient Synopses

Consider a relational table R with d attributes $X_1, X_2, \ldots X_d$. The information in R can be accurately captured as a d-dimensional array A_R, whose j^{th} dimension is indexed by the values of attribute X_j and whose cells contain the count of tuples in R having the corresponding combination of attribute values. (A_R is essentially the *joint frequency distribution* of all the attributes of R.) We have developed a novel, I/O-efficient algorithm for constructing the nonstandard multi-dimensional wavelet decomposition of A_R (denoted W_R). Note that, even though our algorithm computes the decomposition of A_R, it in fact works off the "set-of-tuples" (ROLAP) representation of R. (As noted by Vitter and Wang [17], this is a requirement for computational efficiency since the joint frequency array A_R is typically extremely sparse.) The worst-case I/O complexity of our algorithm matches that of the best algorithms of Vitter and Wang [17], requiring only a logarithmic number of passes over the data. Furthermore, there exist scenarios (e.g., when R is stored in *chunks* [4, 15]) under which our decomposition algorithm can work in a *single pass* over R.

In a nutshell, our algorithm is based on the recursive application of the following key property the nonstandard Haar decomposition: *The decomposition of a d-dimensional array A_R can be computed by $\underline{\text{independently}}$ computing the decomposition for each of the 2^d d-dimensional subarrays corresponding to A_R's quadrants and then performing pairwise averaging and differencing on the computed 2^d averages of A_R's quadrants.* Abstractly, our decomposition algorithm exploits this observation by working in a "depth-first" fashion on d-dimensional chunks of A_R – all the computation required for decomposing a chunk is executed the first time that chunk is loaded into memory.

The size of the wavelet-coefficient synopsis of R is controlled by a *thresholding scheme* that retains only the largest coefficients in absolute normalized value. (This thresholding rule is in fact *provably optimal* with respect to minimizing the overall mean squared error in the data compression [16].) We have also proposed a time- and space-efficient algorithm for *rendering* (i.e., expanding) a synopsis into an approximate "set-of tuples" relation. Due to space constraints, we have chosen to omit the presentation of our wavelet-decomposition and coefficient-rendering algorithms; the details can be found in the full version of this paper [2]. In the remainder of this section, we summarize the notational conventions used throughout the paper.

Notation. Let $\mathcal{D} = \{D_1, D_2, \ldots, D_d\}$ denote the set of dimensions of A_R, where dimension D_j corresponds to the *value domain* of attribute X_j. Without loss of generality, we assume that each dimension D_j is indexed by the set of integers $\{0, 1, \cdots, |D_j| - 1\}$, where $|D_j|$ denotes the size of dimension D_j. Table 1 outlines the notation used in this paper with a brief description of its semantics.

Most of the notation pertaining to wavelet coeffi-

Symbol	Semantics
d	Dimensionality of input relation
R, A_R	Relation and corresponding joint frequency array
X_j , D_j	j^{th} attribute of R and corresponding domain of values ($1 \leq j \leq d$)
$\mathcal{D} = \{D_1, \ldots, D_d\}$	Set of data dimensions of A_R
$W.R.bound[j].\{lo, hi\}$	Support-rectangle boundaries along dimension D_j for W ($1 \leq j \leq d$)
$W.S.sign[j].\{lo, hi\}$	Sign-vector information along dimension D_j for W ($1 \leq j \leq d$)
$W.S.schg[j]$	Sign-change value along dimension D_j for coefficient W ($1 \leq j \leq d$)
$W.v$	Scalar magnitude of coefficient W

Table 1. Notation

cients W has already been described in Section 2.1. The only exception is the *sign-change value vector* $W.S.schg[j]$ that captures the value along dimension j (between $W.R.bound[j].lo$ and $W.R.bound[j].hi$) at which a transition in the value of the sign vector $W.S.sign[j]$ occurs, for each $1 \leq j \leq d$. That is, the sign $W.S.sign[j].lo$ ($W.S.sign[j].hi$) applies to the range $[W.R.bound[j].lo, \ldots, W.S.schg[j] - 1]$ (resp., $[W.S.schg[j], \ldots, W.R.bound[j].hi]$). As a convention, we set $W.S.schg[j]$ equal to $W.R.bound[j].lo$ when there is no "true" sign change along dimension j, i.e., $W.S.sign[j]$ contains $[+,+]$ or $[-,-]$. Note that, for base Haar coefficients with a true sign change along dimension j, $W.S.schg[j]$ is simply the midpoint between $W.R.bound[j].lo$ and $W.R.bound[j].hi$ (Figure 2). This property, however, no longer holds when arbitrary selections and joins are executed over the wavelet coefficients. As a consequence, we need to store sign-change values explicitly in order to support general query processing operations in an efficient manner.

For the remainder of the paper, we use the symbol W_R to denote the set of coefficients *retained (after thresholding)* from the decomposition of R (i.e., the *wavelet-coefficient synopsis* of R).

3 Processing Relational Queries in the Wavelet-Coefficient Domain

In this section, we propose a novel query algebra for wavelet-coefficient synopses. The basic operators of our algebra correspond directly to conventional relational algebra and SQL operators, including the (non-aggregate) `select`, `project`, and `join`, as well as aggregate operators like `count`, `sum`, and `average`. There is, however, one crucial difference: our operators are defined *over the wavelet-coefficient domain*; that is, their input(s) and output are *sets of wavelet coefficients* (rather than relational tables). The motivation for defining a query algebra for wavelet coefficients comes directly from the need for efficient approximate query processing. To see this, consider an n-ary relational query Q over $R_1, \ldots, R_n$ and assume that each relation R_i has been reduced to a (truncated) set of wavelet coefficients W_{R_i}. A simplistic way of processing Q would be to render each synopsis W_{R_i} into the corresponding approximate relation (denoted $\texttt{render}(W_{R_i})$) and process the relational operators in Q over the resulting sets of tuples. This strategy, however, is clearly inefficient: the approximate relation $\texttt{render}(W_{R_i})$ may contain just as many tuples as the original R_i itself, which implies that query execution costs may also be just as high as those of the original query. Therefore, such a "render-then-process" strategy essentially defeats one of the main motivations behind approximate query processing.

On the other hand, the synopsis W_{R_i} is a highly-compressed representation of $\texttt{render}(W_{R_i})$ that is typically orders of magnitude smaller than R_i. Executing Q in the compressed wavelet-coefficient domain can offer tremendous speedups in query execution cost. We therefore define the operators `op` of our query processing algebra over wavelet-coefficient synopses, while guaranteeing the valid semantics depicted pictorially in the transition diagram of Figure 3. (These semantics can be translated to the equivalence $\texttt{render}(\texttt{op}(T_1, \ldots, T_k)) \equiv \texttt{op}(\texttt{render}(T_1, \ldots, T_k))$, for each operator `op`.) Our algebra allows the fast execution of any relational query Q *entirely* over the wavelet-coefficient domain, while guaranteeing that the final (rendered) result is identical to that obtained by executing Q on the approximate input relations.

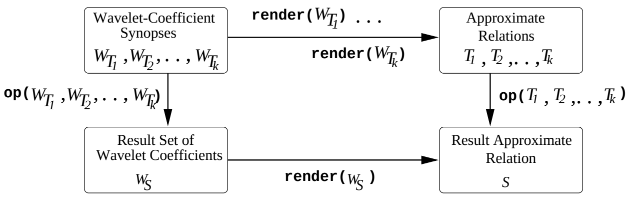

Wavelet-Coefficient Synopses
$W_{T_1}, W_{T_2}, \ldots, W_{T_k}$
render(W_{T_1}) . . .
render(W_{T_k})
Approximate Relations
$T_1, T_2, \ldots, T_k$
op($W_{T_1}, W_{T_2}, \ldots, W_{T_k}$)
op($T_1, T_2, \ldots, T_k$)
Result Set of Wavelet Coefficients
W_S
render(W_S)
Result Approximate Relation
S

Figure 3. Valid semantics for processing query operators over the wavelet-coefficient domain. ($T_1, \ldots, T_k$ can be base relations or intermediate query results.)

In the following subsections, we describe our algorithms for processing `select`, `project`, and `join` operators in the wavelet-coefficient domain. (Our algorithms for processing aggregate operators can be found in [2].) Each operator takes as input one or more set(s) of multi-dimensional wavelet coefficients and appropriately combines and/or updates the components (i.e., hyper-rectangle, sign information, and magnitude) of these coefficients to produce a "valid" set of output coefficients (Figure 3). Note that, while the wavelet coefficients (generated by our decomposition algorithm) for base relational tables have a very regular structure, the same is not necessarily true for the set of coefficients output by an arbitrary `select` or `join` operator. Nevertheless, we loosely continue to refer to the intermediate results of our algebra operators as "wavelet coefficients" since they are characterized by the exact same components as base-relation coefficients (e.g., hyper-rectangle, sign-vectors) and maintain the exact same semantics with respect to the underlying intermediate relation (i.e., the rendering process remains unchanged).

3.1 Selection Operator (`select`)

Our selection operator has the general form $\texttt{select}_{pred}(W_T)$, where $pred$ represents a generic conjunctive predicate on a subset of the d attributes in T; that is, $pred = (l_{i_1} \le X_{i_1} \le h_{i_1}) \wedge \ldots \wedge (l_{i_k} \le X_{i_k} \le h_{i_k})$, where l_{i_j} and h_{i_j} denote the low and high boundaries of the selected range along each selection dimension D_{i_j}, $j = 1, 2, \cdots, k$, $k \le d$. This is essentially a k-dimensional range selection, where the queried range is specified along k dimensions $\mathcal{D}' = \{D_{i_1}, D_{i_2}, \ldots, D_{i_k}\}$ and left unspecified along the remaining $(d - k)$ dimensions $(\mathcal{D} - \mathcal{D}')$. ($\mathcal{D} = \{D_1, D_2, \ldots, D_d\}$ denotes the set of all dimensions of T.) Thus, for each unspecified dimension D_j, the selection range spans the full index domain along the dimension; that is, $l_j = 0$ and $h_j = |D_j| - 1$, for each $D_j \in (\mathcal{D} - \mathcal{D}')$.

The `select` operator effectively filters out the portions of the wavelet coefficients in the synopsis W_T that do not overlap with the k-dimensional selection range, and thus do not contribute to cells in the selected hyper-rectangle. This process is illustrated pictorially in Figure 4(a). More formally, let $W \in W_T$ denote any wavelet coefficient in the input set of our `select` operator. Our approximate query execution engine processes the selection over W as follows. If W's support hyper-rectangle $W.R$ overlaps the k-dimensional selection hyper-rectangle; that is, if *for every* dimension $D_{i_j} \in \mathcal{D}'$, the following condition is satisfied: $l_{i_j} \le W.R.bound[i_j].lo \le h_{i_j}$ or $W.R.bound[i_j].lo \le l_{i_j} \le W.R.bound[i_j].hi$, then

```
1. For all dimensions $D_{i_j} \in \mathcal{D}'$ do
    1.1. Set $W.R.bound[i_j].lo := \max\{l_{i_j}, W.R.bound[i_j].lo\}$ and $W.R.bound[i_j].hi := \min\{h_{i_j}, W.R.bound[i_j].hi\}$.
    1.2. If $W.R.bound[i_j].hi < W.S.schg[i_j]$ then set $W.S.schg[i_j] := W.R.bound[i_j].lo$ and $W.S.sign[i_j] := [W.S.sign[i_j].lo, W.S.sign[i_j].lo]$.
    1.3. Else if $W.R.bound[i_j].lo \ge W.S.schg[i_j]$ then set $W.S.schg[i_j] := W.R.bound[i_j].lo$ and $W.S.sign[i_j] := [W.S.sign[i_j].hi, W.S.sign[i_j].hi]$.
2. Add the (updated) $W$ to the set of output coefficients; that is, set $W_S := W_S \cup \{W\}$, where $S = \texttt{select}_{pred}(T)$.
```

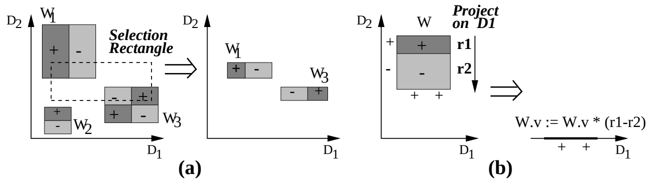


Figure 4. Processing (a) selection and (b) projection operations in the wavelet-coefficient domain.

Our `select` processing algorithm chooses (and appropriately updates) only the coefficients in W_T that overlap with the k-dimensional selection hyper-rectangle. For each such coefficient, our algorithm (a) updates the hyper-rectangle boundaries according to the specified selection range (Step 1.1), and (b) updates the sign information, if such an update is necessary (Steps 1.2-1.3). Briefly, the sign information along the queried dimension D_{i_j} needs to be updated only if the selection range along D_{i_j} is completely contained in either the low (1.2) or the high (1.3) sign-vector range of the coefficient along D_{i_j}. In both cases, the sign-vector of the coefficient is updated to contain only the single sign present in the selection range and the coefficient's sign-change is set to its leftmost boundary value (since there is no change of sign along D_{i_j} after the selection). The sign-vector and sign-change of the result coefficient remain untouched (i.e., identical to those of the input coefficient) if the selection range spans the original sign-change value.

3.2 Projection Operator (`project`)

Our projection operator has the general form $\texttt{project}_{X_{i_1},\ldots,X_{i_k}}(W_T)$, where the k projection attributes $X_{i_1}, \ldots, X_{i_k}$ form a subset of the d attributes of T. Letting $\mathcal{D}' = \{D_{i_1}, \ldots, D_{i_k}\}$ denote the $k \le d$ projection dimensions, we are interested in *projecting out* the $d - k$ dimensions in $(\mathcal{D} - \mathcal{D}')$. We give a general method for projecting out a single dimension $D_j \in \mathcal{D} - D'$. This method can then be applied repeatedly to project out all the dimensions in $(\mathcal{D} - \mathcal{D}')$, one dimension at a time.

Consider T's corresponding multi-dimensional array A_T. Projecting a dimension D_j out of A_T is equivalent to summing up the counts for all the array cells in each one-dimensional row of A_T along dimension D_j and then assigning this aggregated count to the single cell corresponding to that row in the remaining dimensions $(\mathcal{D} - \{D_j\})$. Consider any d-dimensional wavelet coefficient W in the `project` operator's input set W_T. Remember that W contributes a value of $W.v$ to every cell in its support hyper-rectangle $W.R$. Furthermore, the sign of this contribution for every one-dimensional row along dimension D_j is determined as either $W.S.sign[j].hi$ (if the cell lies above $W.S.schg[j]$) or $W.S.sign[j].lo$ (otherwise). Thus, we can work directly on the coefficient W to project out dimension D_j by simply adjusting the coefficient's magnitude with an appropriate multiplicative constant $W.v := W.v * p_j$, where p_j is defined as:

$$(W.R.bound[j].hi - W.S.schg[j] + 1) * W.S.sign[j].hi \ + \ (W.S.schg[j] - W.R.bound[j].lo) * W.S.sign[j].lo. \quad (1)$$

A two-dimensional example of projecting out a dimension in the wavelet-coefficient domain is depicted in Figure 4(b). Multiplying $W.v$ with p_j (Equation (1)) effectively projects out dimension D_j from W by summing up W's contribution on each one-dimensional row along dimension D_j. Of course, besides adjusting $W.v$, we also need to discard dimension D_j from the hyper-rectangle and sign information for W, since it is now a $(d-1)$-dimensional coefficient (on dimensions $\mathcal{D} - \{D_j\}$). Note that if the coefficient's sign-change lies in the middle of its support range along dimension D_j (e.g., see Figure 2(a)), then its adjusted magnitude will be 0, which means that it can safely be discarded from the output set of the projection operation.

Repeating the above process for each wavelet coefficient $W \in W_T$ and each dimension $D_j \in \mathcal{D} - D'$ gives the set of output wavelet coefficients W_S, where $S = \texttt{project}_{\mathcal{D}'}(T)$. Equivalently, given a coefficient W, we can simply set $W.v := W.v * \prod_{D_j \in \mathcal{D}-D'} p_j$ (where p_j is as defined in Equation (1)) and discard dimensions $\mathcal{D} - D'$ from W's representation.

3.3 Join Operator (join)

Our join operator has the general form $\texttt{join}_{pred}(W_{T_1}, W_{T_2})$, where T_1 and T_2 are (approximate) relations of arity d_1 and d_2, respectively, and $pred$ is a conjunctive k-ary equi-join predicate of the form $(X_1^1 = X_1^2) \wedge \ldots \wedge (X_k^1 = X_k^2)$, where X_j^i (D_j^i) $(j = 1, \ldots, d_i)$ denotes the j^{th} attribute (resp., dimension) of T_i $(i = 1, 2)$. (Without loss of generality, we assume that the join attributes are the first $k \leq \min\{d_1, d_2\}$ attributes of each joining relation.) Note that the result of the join operation W_S is a set of $(d_1 + d_2 - k)$-dimensional wavelet coefficients; that is, the join operation returns coefficients of (possibly) different arity than any of its inputs.

To see how our join processing algorithm works, consider the multi-dimensional arrays A_{T_1} and A_{T_2} corresponding to the join operator's input arguments. Let $(i_1^1, \ldots, i_{d_1}^1)$ and $(i_1^2, \ldots, i_{d_2}^2)$ denote the coordinates of two cells belonging to A_{T_1} and A_{T_2}, respectively. If the indexes of the two cells match on the join dimensions, i.e., $i_1^1 = i_1^2, \ldots, i_k^1 = i_k^2$, then the cell in the join result array A_S with coordinates $(i_1^1, \ldots, i_{d_1}^1, i_{k+1}^2, \ldots, i_{d_2}^2)$ is populated with the *product* of the count values contained in the two joined cells. Since the cell counts for A_{T_i} are derived by appropriately summing the contributions of the wavelet coefficients in W_{T_i} and, of course, a numeric product can always be distributed over summation, we can process the join operator entirely in the wavelet-coefficient domain by considering all pairs of coefficients from W_{T_1} and W_{T_2}. Briefly, for any two coefficients from W_{T_1} and W_{T_2} that overlap in the join dimensions and, therefore, contribute to joining data cells, we define an output coefficient with magnitude equal to the product of the two joining coefficients and a support hyper-rectangle with ranges that are (a) equal to the overlap of the two coefficients for the k (common) join dimensions, and (b) equal to the original coefficient ranges along any of the $d_1 + d_2 - 2k$ remaining dimensions. The sign information for an output coefficient along any of the k join dimensions is derived by appropriately multiplying the sign-vectors of the joining coefficients along that dimension, taking care to ensure that only signs along the overlapping portion are taken into account. (The sign information along non-join dimensions remains unchanged.) An example of this process in two dimensions ($d_1 = d_2 = 2$, $k = 1$) is depicted in Figure 5(a).

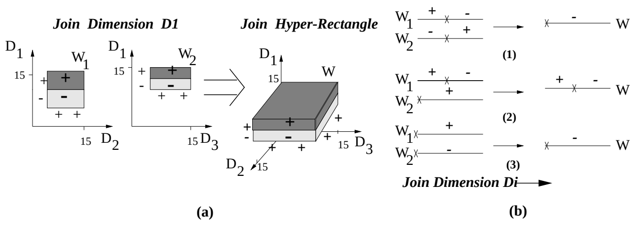


Figure 5. (a) Processing join operations in the wavelet-coefficient domain. (b) Computing sign information for join output coefficients.

More formally, our approximate query execution strategy for joins can be described as follows. (To simplify the notation, we ignore the "1/2" superscripts and denote the join dimensions as $D_1, \ldots, D_k$, and the remaining $d_1 + d_2 - 2k$ dimensions as $D_{k+1}, \ldots, D_{d_1+d_2-k}$.) For each pair of wavelet coefficients $W_1 \in W_{T_1}$ and $W_2 \in W_{T_2}$, if the coefficients' support hyper-rectangles overlap in the k join dimensions; that is, if *for every* dimension D_i, $i = 1 \ldots, k$, the following condition is satisfied:

$$W_1.R.bound.lo[i] \leq W_2.R.bound.lo[i] \leq W_1.R.bound.hi[i] \quad \text{or}$$
$$W_2.R.bound.lo[i] \leq W_1.R.bound.lo[i] \leq W_2.R.bound.hi[i],$$

then the corresponding output coefficient $W \in W_S$ is defined in the following steps.

```
1. For all join dimensions $D_i$, $i = 1, \ldots, k$ do
   1.1. Set $W.R.bound[i].lo := \max\{W_1.R.bound[i].lo, W_2.R.bound[i].lo\}$ and $W.R.bound[i].hi := \min\{W_1.R.bound[i].hi, W_2.R.bound[i].hi\}$.
   1.2. For $j = 1, 2$
            /* let $s_j$ be a temporary sign-vector variable */
      1.2.1. If $W.R.bound[i].hi < W_j.S.schg[i]$ then set
             $s_j := [W_j.S.sign[i].lo, W_j.S.sign[i].lo]$.
      1.2.2. Else if $W.R.bound[i].lo \geq W_j.S.schg[i]$ then
             set $s_j := [W_j.S.sign[i].hi, W_j.S.sign[i].hi]$.
      1.2.3. Else set $s_j := W_j.S.sign[i]$.
   1.3. Set $W.S.sign[i] := [s_1.lo * s_2.lo\ ,\ s_1.hi * s_2.hi]$.
   1.4. If $W.S.sign[i].lo == W.S.sign[i].hi$ then set
        $W.S.schg[i] := W.R.bound[i].lo$.
   1.5 Else set $W.S.schg[i] := \max_{j=1,2}\{W_j.S.schg[i] : W_j.S.schg[i] \in [W.R.bound[i].lo, W.R.bound[i].hi]\}$.
2. For each (non-join) dimension $D_i$, $i = k+1, \ldots, d_1$ do:
   Set $W.R.bound[i] := W_1.R.bound[i]$, $W.S.sign[i] := W_1.S.sign[i]$ , and $W.S.schg[i] := W_1.S.schg[i]$.
3. For each (non-join) dimension $D_i$, $i = d_1+1, \ldots, d_1+d_2-k$ do: Set $W.R.bound[i] := W_2.R.bound[i-d_1+k]$, $W.S.sign[i] := W_2.S.sign[i-d_1+k]$ , and
   $W.S.schg[i] := W_2.S.schg[i-d_1+k]$.
4. Set $W.v := W_1.v * W_2.v$ and $W_S := W_S \cup \{W\}$,
   where $S = \texttt{join}_{pred}(T_1, T_2)$.
```

Note that the bulk of our join processing algorithm concentrates on the correct settings for the output coefficient W along the k join dimensions (Step 1), since the problem becomes trivial for the $d_1 + d_2 - k$ remaining dimensions (Steps 2-3). Given a pair of joining input coefficients and a join dimension D_i, our algorithm starts out by setting the hyper-rectangle range of the output coefficient W along D_i equal to the overlap of the two input coefficients along D_i (Step 1.1). We then proceed to compute W's sign information along join dimension D_i (Steps 1.2-1.3) , which is slightly more involved. (Remember that T_1 and T_2 are (possibly) the results of earlier `select` and/or `join` operators, which means that their rectangle boundaries and signs along D_i can be arbitrary.) The basic idea is to determine, for each of the two input coefficients W_1 and W_2, where the boundaries of the join range lie with respect to the coefficient's sign-change value along dimension D_i. Given an input coefficient W_j ($j = 1, 2$), if the join range along D_i is completely contained in either the low (1.2.1) or the high (1.2.2) sign-vector range of W_j along D_i, then a temporary sign-vector s_j is appropriately set (with the same sign in both entries). Otherwise, i.e., if the join range spans W_j's sign-change (1.2.3), then s_j is simply set to W_j's sign-vector along D_i. Thus, s_j captures the sign of coefficient W_j in the joining range, and multiplying s_1 and s_2 (element-wise) yields the sign-vector for the output coefficient W along dimension D_i (Step 1.3). If the resulting sign vector for W does not contain a true sign change (i.e., the low and high components of $W.S.sign[i]$ are the same), then W's sign-change value along dimension D_i is set equal to the low boundary of $W.R$ along D_i, according to our convention (Step 1.4). Otherwise, the sign-change value for the output coefficient W along D_i is set equal to the maximum of the input coefficients' sign-change values that are contained in the join range (i.e., $W.R$'s boundaries) along D_i (Step 1.5).

In Figure 5(b), we illustrate three common scenarios for the computation of W's sign information along the join dimension D_i. The left-hand side of the figure shows three possibilities for the sign information of the input coefficients W_1 and W_2 along the join range of dimension D_i (with crosses denoting sign changes). The right-hand side depicts the resulting sign information for the output coefficient W along the same range. The important thing to observe with respect to our sign-information computation in Steps 1.3–1.5 is that the join range along any join dimension D_i can contain *at most one* true sign change. By this, we mean that if the sign for input coefficient W_j actually changes in the join range along D_i, then this sign-change value is unique; that is, the two input coefficients cannot have true sign changes at distinct points of the join range. This follows from the *complete containment* property of the base coefficient ranges along dimension D_i (Section 2.1). (Note that our algorithm for `select` retains the value of a true sign change for a base coefficient if it is contained in the selection range, and sets it equal to the value of the left boundary otherwise.) This range containment along D_i ensures that if W_1 and W_2 both contain a true sign change in the join range (i.e., their overlap) along D_i, then that will occur *at exactly the same value* for both (as illustrated in Figure 5(b.1)). Thus, in Step 1.3, W_1's and W_2's sign vectors in the join range can be multiplied to derive W's sign-vector. If, on the other hand, one of W_1 and W_2 has a true sign change in the join range (as shown in Figure 5(b.2)), then the max operation of Step 1.5 will always set the sign change of W along D_i correctly to the true sign-change value (since the other sign change will either be at the left boundary or outside the join range). Finally, if neither W_1 nor W_2 have a true sign change in the join range, then the high and low components of W's sign vector will be identical and Step 1.4 will set W's sign-change value correctly.

Example 3.1: Consider the wavelet coefficients W_1 and W_2 in Figure 5. Let the boundaries and sign information of W_1 and W_2 along the join dimension D_1 be as follows: $W_1.R.bound[1] = [4, 15]$, $W_2.R.bound[1] = [8, 15]$, $W_1.S.sign[1] = [-, +]$, $W_2.S.sign[1] = [-, +]$, $W_1.S.schg[1] = 8$, and $W_2.S.schg[1] = 12$. In the following, we illustrate the computation of the hyper-rectangle and sign information for join dimension D_1 for the coefficient W that is output by our algorithm when W_1 and W_2 are "joined". Note that for the non-join dimensions D_2 and D_3, this information for W is identical to that of W_1 and W_2 (respectively), so we focus solely on the join dimension D_1.

First, in Step 1.1, $W.R.bound[1]$ is set to $[8, 15]$, i.e., the overlap range between W_1 and W_2 along D_1. In Step 1.2.2, since $W.R.bound[1].lo = 8$ is greater than or equal to $W_1.S.schg[1] = 8$, we set $s_1 = [+, +]$. In Step 1.2.3, since $W_2.S.schg[1] = 12$ lies in between $W.R$'s boundaries, we set $s_2 = [-, +]$. Thus, in Step 1.3, $W.S.sign[1]$ is set to the product of s_1 and s_2 which is $[-, +]$. Finally, in Step 1.5, $W.S.schg[1]$ is set to the maximum of the sign change values for W_1 and W_2 along dimension D_1, or $W.S.schg[1] := \max\{8, 12\} = 12$. ∎

4 Experimental Study

In this section, we present the results of an extensive empirical study that we have conducted using the novel query processing tools developed in this paper. The objective of this study is twofold: (1) to establish the effectiveness of our wavelet-based approach to approximate query processing, and (2) to demonstrate the benefits of our methodology compared to earlier approaches based on sampling and histograms. Our experiments consider a wide range of queries executed on both synthetic and real-life data sets. The major findings of our study can be summarized as follows.

• **Improved Answer Quality.** The quality/accuracy of the approximate answers obtained from our wavelet-based query processor is, in general, better than that obtained by either sampling or histograms for a wide range of data sets and `select`, `project`, `join`, and aggregate queries.

• **Low Synopsis Construction Costs.** Our I/O-efficient wavelet decomposition algorithm is extremely fast and scales linearly with the size of the data (i.e., the number of cells in the MOLAP array). In contrast, histogram construction costs increase explosively with the dimensionality of the data.

• **Fast Query Execution.** Query execution-time speedups of more than two orders of magnitude are made possible by our approximate query processing algorithms. Furthermore, our query execution times are competitive with those obtained by the histogram-based methods of Ioannidis and Poosala [6], and sometimes significantly faster (e.g., for `join`s).

Thus, our experimental results validate the thesis of this paper that wavelets are a viable, effective tool for general-purpose approximate query processing in DSS environments. All experiments reported in this section were performed on a Sun Ultra-2/200 machine with 512 MB of main memory, running Solaris 2.5.

Due to space constraints, the presentation in this paper focuses on the results of our experimental evaluation with real-life data sets, which are indicative of the overall set of experimental results. The details of our experimentation with synthetic data sets can be found in the full version of this paper [2].

4.1 Experimental Testbed and Methodology

Techniques. We consider three approximate query answering techniques in our study.

• *Sampling.* A random sample of the non-zero cells in the multi-dimensional array representation for each base relation is selected , and the counts for the cells are appropriately scaled. Thus, if the total count of all cells in the array is t and the sum of the counts of cells in the sample is s, then the count of every cell in the sample is multiplied by $\frac{t*v}{s}$. These scaled counts give the tuple counts for the corresponding approximate relation.

• *Histograms.* Each base relation is approximated by a multi-dimensional MaxDiff(V,A) histogram. Our choice of this histogram class is motivated by the recent work of Ioannidis and Poosala [6], where it is shown that MaxDiff(V,A) histograms result in higher-quality approximate query answers compared to other histogram classes. We process `select`s, `join`s, and aggregate operators on histograms as described in [6]. For instance, while `select`s are applied directly to the histogram for a relation, a `join` between two relations is done by first partially expanding their histograms to generate the tuple-value distribution of the each relation. An indexed nested-loop `join` is then performed on the resulting tuples.

• *Wavelets.* Wavelet-coefficient synopses are constructed on the base relations (using our decomposition algorithm) and query processing is performed entirely in the wavelet-coefficient domain, as described in Section 3. In our `join` implementation, overlapping pairs of coefficients are determined using a simple nested-loop join. Further, during the rendering step for non-aggregate queries, cells with negative counts are not included in the final answer to the query.

Since we assume d dimensions in the multi-dimensional array for a d-attribute relation, c random samples require $c*(d+1)$ units of space; d units are needed to store the index of the cell and 1 unit is required to store the cell count. Storing c wavelet coefficients also requires the same amount of space, since we need d units to specify the position of the coefficient in the wavelet transform array and 1 unit to specify the value for the coefficient. (Note that the hyper-rectangle and sign information for a base coefficient can easily be derived from its location in the wavelet transform array.) On the other hand, each histogram bucket requires $3*d+1$ units of space; $2*d$ units to specify the low and high boundaries for the bucket along each of the d dimensions, d units to specify the number of distinct values along each dimension, and 1 unit to specify the average frequency for the bucket [12]. Thus, for a given amount of space corresponding to c samples/wavelet coefficients, we store $b \approx \frac{c}{3}$ histogram buckets to ensure a fair comparison between the methods.

Queries. The workload used to evaluate the various approximation techniques consists of four main query types: (1) `SELECT` *Queries*: ranges are specified for (a subset of) the attributes in a relation and all tuples that satisfy the conjunctive range predicate are returned as part of the query result, (2) `SELECT-SUM` *Queries*: the total `sum` of a particular attribute's values is computed for all tuples that satisfy a conjunctive range predicate over (a subset of) the attributes, (3) `SELECT-JOIN` *Queries*: after performing selections on two input relations, an equi-join on a single join dimension is performed and the resulting tuples are output; and, (4) `SELECT-JOIN-SUM` *Queries*: the total `sum` of an attribute's values is computed over all the tuples resulting from a `SELECT-JOIN`.

For each of the above query types, we have conducted experiments with multiple different choices for (a) `select` ranges, and (b) `select`, `join`, and `sum` attributes. The results presented in the next section are indicative of the overall observed behavior of the schemes. Furthermore, the queries presented in this paper are fairly representative of typical queries over our data sets.

Answer-Quality Metrics. In our experiments with aggregate queries (e.g., `SELECT-SUM` queries), we use the *absolute relative error* in the aggregate value as a measure of the accuracy of the approximate query answer. When deciding on an error metric for non-aggregate results, we considered both the *Match And Compare* (MAC) error of Ioannidis and Poosala [6] and the network-flow-based *Earth Mover's Distance* (EMD) error of Rubner et al. [14]. We eventually chose a variant of the EMD error metric, since it offers a number of advantages over MAC error (e.g., computational efficiency, natural handling of non-integral counts) and, furthermore, we found that MAC error can show unstable behavior under certain circumstances. A more detailed discussion on MAC and EMD errors as well as the actual EMD error calculation formulas used in this paper can be found in the full paper [2].

4.2 Query Execution Times

In order to compare the query processing times for the various approaches, we measured the time (in seconds) for executing a SELECT-JOIN-SUM query over synopses of two-dimensional synthetic data sets using each approach. We do not consider the time for random sampling since the join results with samples did not generate any tuples, except for very large sample sizes. The running time of the join query on the original base relations (using an indexed nested-loop join) to produce an exact answer was 3.6 seconds. In practice, we expect that this time will be much higher since in our case, the entire relations fit in main memory. As is evident from Table 2, our wavelet-based technique is more than two orders of magnitude faster compared to running the queries on the entire base relations.

Technique	**Number of Coefficients**			
	500	1000	2000	5000
Wavelets	0.01	0.02	0.04	0.08
Histograms	9.8	1.48	0.43	1.26

Table 2. SELECT-JOIN-SUM Query Execution Times

Also, note that the performance of histograms is much worse than that of wavelets. The explanation lies in the fact that the join processing algorithm of Ioannidis and Poosala [6] requires joining histograms to be partially expanded to generate the tuple-value distribution for the corresponding approximate relations. The problem with this approach is that the intermediate relations can become fairly large and may even contain more tuples than the original relations. For example, with 500 coefficients, the expanded histogram contains almost 5 times as many tuples as the base relations. The sizes of the approximate relations decrease as the number of buckets increase, and thus execution times for histograms drop for larger numbers of buckets. In contrast, in our wavelet approach, join processing is carried out exclusively in the compressed domain, that is, joins are performed directly on the wavelet coefficients without ever materializing intermediate relations. The tuples in the final query answer are generated at the very end as part of the rendering step and this is the primary reason for the superior performance of the wavelet approach.

4.3 Experimental Results – Real-life Data Sets

We obtained our real-life data set from the US Census Bureau (www.census.gov). We employed the Current Population Survey (CPS) data source and within it the Person Data Files of the March Questionnaire Supplement. We used the 1992 data file for the select and select sum queries, and the 1992 and 1994 data files for the join and join sum queries. For both files, we projected the data on the following four attributes whose domain values were previously coded: *age* (with value domain 0 to 17), *educational attainment* (with value domain 0 to 46), *income* (with value domain 0 to 41) and *hours per week* (with value domain 0 to 13). Along with each tuple in the projection, we stored a count which is the number of times it appears in the file. We rounded the maximum domain values off to the nearest power of 2 resulting in domain sizes of 32, 64, 64 and 16 for the four dimensions, and a total of 2 million cells in the

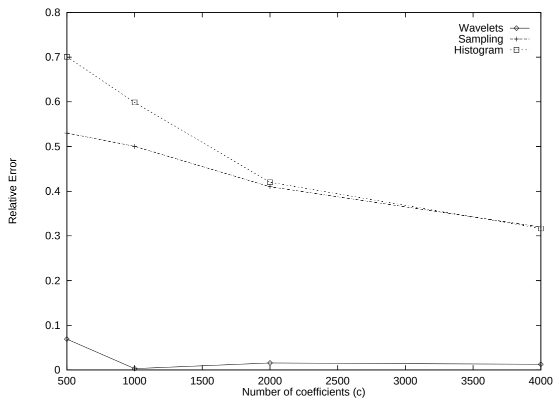


Figure 7. SELECT-JOIN-SUM queries on real-life data.

array. The 1992 and the 1994 collections had 16271 and 16024 cells with non-zero counts, respectively, resulting in a density of ≈ 0.001. However, even though the density is very low, we did observe large dense regions within the arrays when we visualized the data – these dense regions spanned the entire domains of the *age* and *income* dimensions.

For all the queries, we used the following select range: $5 \leq age < 10$ and $10 \leq income < 15$ that we found to be representative of several select ranges that we considered (the remaining two dimensions were left unspecified). The selectivity of the query was 1056/16271= 6%. For sum queries, the sum operation was performed on the *age* dimension. For join queries, the join was performed on the *age* dimension between the 1992 and 1994 data files.

SELECT Queries. In Figures 6(a) and 6(b), we plot the EMD error and relative error for SELECT and SELECT-SUM queries, respectively, as the space allocated for the approximations is increased from 3% to 25% of the relation. From the graphs, it follows that wavelets result in the least value for the EMD error, while sampling has the highest EMD error. For SELECT-SUM queries, wavelets exhibit more than an order of magnitude improvement in relative error compared to both histograms and sampling (the relative error for wavelets is between 0.5% and 3%). Thus, the results for the select queries indicate that wavelets are effective at accurately capturing both the value as well as the frequency distribution of the underlying real-life data set.

It is interesting to note that the relative error for sampling is better than that of histograms. We conjecture that one of the reasons for this is the higher dimensionality of the real-life data sets, where histograms are less effective.

JOIN Queries. We only plot the results of the SELECT-JOIN-SUM queries in Figure 7, since the EMD error graphs for SELECT-JOIN queries were similar. Over the entire range of coefficients, wavelets outperform sampling and histograms, in most cases by more than an order of magnitude. With the real-life data set, even after the join, the relative aggregate error using wavelets is very low and ranges between 1% to 6%. The relative error of all

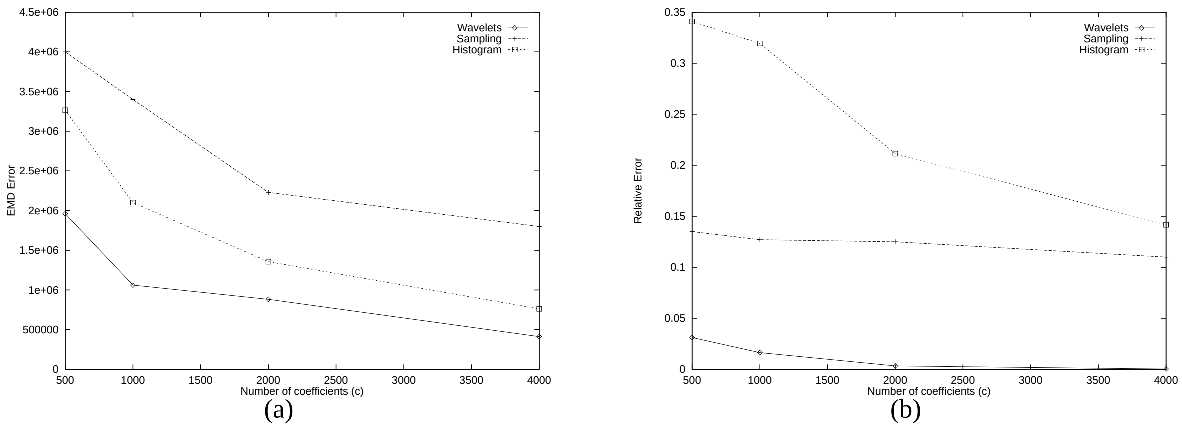


Figure 6. (a) `SELECT` and (b) `SELECT-SUM` query errors on real-life data.

the techniques improve as the amount of allocated space is increased. Compared to the synthetic data sets, where the result of a `join` over samples contained zero tuples in most cases, for the real-life data sets, sampling performs quite well. This is because the size of the domain of the *age* attribute on which the join is performed is only 18, which is quite small. Consequently, the result of the `join` query over the samples is no longer empty.

5 Conclusions

In this paper, we have proposed the use of multi-dimensional wavelets as an effective tool for general-purpose approximate query processing in modern, high-dimensional applications. Our approach is based on building wavelet-coefficient synopses of the data and using these synopses to provide approximate answers to queries. We have developed novel query processing algorithms that operate directly on the wavelet-coefficient synopses of relational data, thus allowing for very fast processing of arbitrarily complex queries *entirely* in the wavelet-coefficient domain. We have also proposed a novel I/O-efficient wavelet decomposition algorithm for building the synopses of relational data. Finally, we have conducted an extensive experimental study with synthetic as well as real-life data sets that verifies the effectiveness of our wavelet-based approach compared to both sampling and histograms.

Acknowledgments: We would like to thank Vishy Poosala for providing us with his MaxDiff histogram computation code. This work was partially supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[1] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. “Join Synopses for Approximate Query Answering”. In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*.

[2] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. “Approximate Query Processing Using Wavelets”. February 2000. Bell Labs Tech. Memorandum.

[3] W. G. Cochran. “*Sampling Techniques*”. John Wiley & Sons, 1977. (Third Edition).

[4] P. M. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. “Caching Multidimensional Queries Using Chunks”. In *Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*.

[5] J. M. Hellerstein, P. J. Haas, and H. J. Wang. “Online Aggregation”. In *Proc. of the 1997 ACM SIGMOD Intl. Conf. on Management of Data*.

[6] Y. E. Ioannidis and V. Poosala. “Histogram-Based Approximation of Set-Valued Query Answers”. In *Proc. of the 25th Intl. Conf. on Very Large Data Bases*, 1999.

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

[8] R. J. Lipton, J. F. Naughton, and D. A. Schneider. “Practical Selectivity Estimation through Adaptive Sampling”. In *Proc. of the 1990 ACM SIGMOD Intl. Conf. on Management of Data*.

[9] Y. Matias, J. S. Vitter, and M. Wang. “Wavelet-Based Histograms for Selectivity Estimation”. In *Proc. of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*.

[10] A. Natsev, R. Rastogi, and K. Shim. “WALRUS: A Similarity Retrieval Algorithm for Image Databases”. In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*.

[11] V. Poosala and V. Ganti. “Fast Approximate Answers to Aggregate Queries on a Data Cube”. In *Proc. of the 1999 Intl. Conf. on Scientific and Statistical Database Management*.

[12] V. Poosala and Y. E. Ioannidis. “Selectivity Estimation Without the Attribute Value Independence Assumption”. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, 1997.

[13] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. “Improved Histograms for Selectivity Estimation of Range Predicates”. In *Proc. of the 1996 ACM SIGMOD Intl. Conf. on Management of Data*.

[14] Y. Rubner, C. Tomasi, and L. Guibas. “A Metric for Distributions with Applications to Image Databases”. In *Proc. of the 1998 IEEE Intl. Conf. on Computer Vision*.

[15] S. Sarawagi and M. Stonebraker. “Efficient Organization of Large Multidimensional Arrays”. In *Proc. of the 10th Intl. Conf. on Data Engineering*, 1994.

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

[17] J. S. Vitter and M. Wang. “Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets”. In *Proc. of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*.

[18] J. S. Vitter, M. Wang, and B. Iyer. “Data Cube Approximation and Histograms via Wavelets”. In *Proc. of the 7th Intl. Conf. on Information and Knowledge Management*, 1998.