ProPolyne: A Fast Wavelet-based Algorithm for Progressive Evaluation of Polynomial Range-Sum Queries *

Rolfe R. Schmidt and Cyrus Shahabi

University of Southern California, Los Angeles CA 90089-0781, USA
{rrs, shahabi}@usc.edu

Abstract. Many range aggregate queries can be efficiently derived from a class of fundamental queries: the *polynomial range-sums*. After demonstrating how any range-sum can be evaluated exactly in the wavelet domain, we introduce a novel pre-aggregation method called ProPolyne to evaluate arbitrary polynomial range-sums progressively. At each step of the computation, ProPolyne makes the best possible wavelet approximation of the submitted query. The result is a data-independent approximate query answering technique which uses data structures that can be maintained efficiently and exactly. ProPolyne's performance as an exact algorithm is comparable to the best known MOLAP techniques. Experimental results show that this approach of approximating queries rather than compressing data produces consistent and superior approximate results when compared to typical wavelet-based data compression techniques.

1 Introduction

Range aggregate queries are a fundamental part of modern data analysis applications. Many recently proposed techniques can be used to evaluate a variety of aggregation operators from simple COUNT and SUM queries to statistics such as VARIANCE and COVARIANCE. Most of these methods attempt to provide efficient query answering at a reasonable update cost for a single aggregation operation. An interesting exception is [20], which points out that all second order statistical aggregation functions (including hypothesis testing, principle component analysis, and ANOVA) can be derived from SUM queries of second order polynomials in the measure attributes. These SUM queries can be efficiently supported using any proposed OLAP method. Higher order statistics can similarly be reduced to sums of higher order polynomials. The power of these observations leads us to introduce the class of *polynomial range-sum* aggregates in Section 3.

* This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and ITR-0082826, NASA/JPL contract nr. 961518, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash/equipment gifts from NCR, IBM, Intel and SUN.

There is a generic way to support polynomial range-sums of any degree using existing OLAP techniques- simply treat each independent monomial up to the required degree as a separate measure and build a new datacube for each of these. This requires the measure attributes to be specified when the database is populated, and has unwieldy storage and maintenance cost for higher order polynomials.

We propose a novel MOLAP technique that can support any polynomial range-sum query (up to a degree specified when the database is populated) using a single set of precomputed aggregates. This extra power comes with little extra cost: the query, update, and storage costs are comparable to the best known MOLAP techniques (see Table 1). We achieve this by observing that polynomial range-sums can be translated and evaluated in the wavelet domain. When the wavelets are chosen to satisfy an appropriate *moment condition*, most of the query wavelet coefficients vanish making the query easier to evaluate. We make this observation practical by introducing the *lazy wavelet transform*, an algorithm that translates polynomial range-sums to the wavelet domain in poly-logarithmic time (Section 6).

Wavelets are often thought of as a data approximation tool, and have been used this way for approximate range query answering [7, 21]. The efficacy of this approach is highly data dependent; it only works when the data have a concise wavelet approximation. Furthermore the wavelet approximation is difficult to maintain. To avoid these problems, we use wavelets to approximate incoming queries rather than the underlying data. By using our exact polynomial range-sum technique, but using the largest query wavelet coefficients first we are able to obtain accurate, data-independent query approximations after a small number of I/Os. This approach naturally leads to a progressive algorithm.

We bring these ideas together by introducing ProPolyne (Progressive Polynomial Range-Sum Evaluator, Section 7), a polynomial range-sum evaluation method which

1. Treats all dimensions, including measure dimensions, symmetrically and supports range-sum queries where the measure is *any* polynomial in the data dimensions. All computation is performed entirely in the wavelet domain (see Section 5).
2. Uses the lazy wavelet transform to achieve query and update cost comparable to the best known exact techniques (see Table 1 and Section 6).
3. By using the most important query wavelet coefficients first, provides excellent approximate results with very little I/O and computational overhead, reaching low relative error far more quickly than analogous data compression methods (see Section 7).
4. Provides efficiently computable guaranteed error bounds on all approximate query results (see Section 8).

Experimental results on several empirical datasets show that the approximate results produced by ProPolyne are very accurate long before the exact query evaluation is complete (Section 9). These experiments also show that the

performance of wavelet based data approximation methods varies wildly with the dataset, while *query approximation* based ProPolyne delivers consistent, and consistently better, results.

2 Related Work

Extensive research has been done to find efficient ways to evaluate range aggregates. The prefix-sum method [11] publicized the fact that careful pre-aggregation can be used to evaluate range aggregate queries in time independent of the range size. This led to a number of new techniques that provide similar benefits with different query/update cost tradeoffs [6, 5, 16, 3]. Iterative Data Cubes [17] generalize all of these techniques, and allow different methods to be used on each dimension. It is pointed out in [20] that statistical analysis of summary data can lead to significant errors in applications such as regression rule mining. In order to provide efficient OLAP-style range statistics, Multivariate Aggregate Views are proposed. These provide a ROLAP technique to support polynomial range-sums of degree 2. ProPolyne takes a MOLAP approach to this problem, and is able to use one set of precomputed aggregates to support all polynomial range-sums up to an arbitrary degree.

Approximate query answering can be used to deliver fast query results. Use of synopsis data structures such as histograms has been analyzed thoroughly [13, 7]. References [19, 9, 18] create synopses that estimate the data frequency distribution. The flexible measures supported by this approach inspire its use in this paper. Online aggregation, or progressive query answering, has also been proposed [10, 15, 23, 12]. These methods, whether based on sampling, R-trees, or multiresolution analysis, share a common strategy: answer queries quickly using a low resolution view of the data, and progressively refine the answer while building a sharper view. ProPolyne is fundamentally different. It makes optimal progressive estimates of the query, not the data. Furthermore, the progressive evaluation comes for free since ProPolyne is an excellent exact algorithm.

Recently wavelets have emerged as a powerful tool for approximate answering of aggregate [7, 21, 23] and relational algebra [2] queries. Streaming algorithms for approximate population of a wavelet database are also available [8]. Most of the wavelet query evaluation work has focused on using wavelets to compress the underlying data, reducing the size of the problem. ProPolyne can use compressed data, but is designed to work as an exact algorithm with uncompressed data. ProPolyne produces approximate query results by compressing queries, not data.

3 Range Queries

In this section we introduce the class of *polynomial range-sum* queries, and show how we use the *data frequency distribution* to recast these queries as *vector queries*. Throughout this paper we work with a finite instance I of a schema F (the 'fact table'). We assume that each attribute of F is numeric with domain of size $N = 2^j$, and that F has d attributes. We study the following:

Definition 1. *Given a* range $R \subset \mathrm{Dom}(F)$ *and a* measure function $f : \mathrm{Dom}(F) \to \mathbb{R}$*, the* range-sum query $Q(R, f)|_I$ *is equal to* $Q(R, f)|_I = \sum_{\bar{x} \in I \cap R} f(\bar{x})$ *The sum counts multiplicities in the multiset* I*. If* f *is a polynomial of degree* δ *we call this a* polynomial range-sum *of degree* δ*.*

Example 1. Consider a table of employee ages and salaries, taken from the example in [19], with entries [age, salary]: { [25, 50K], [28, 55K], [30, 58K], [50, 100K], [55, 130K], [57, 120K]} . Choose the range R to be the set of all points where $25 \leq \text{age} \leq 40$ and $55000 \leq \text{salary} \leq 150000$.

By choosing $f(\bar{x}) \equiv \mathbf{1}(\bar{x}) = 1$ the range-sum query returns the COUNT of the tuples that lie in R:

$$Q(R, \mathbf{1}, I) = \sum_{\bar{x} \in R \cap I} \mathbf{1}(\bar{x}) = \mathbf{1}(28, 55K) + \mathbf{1}(30, 58K) = 2$$

Choosing $f(\bar{x}) \equiv \text{salary}(\bar{x})$ the range-sum query computes the SUM of salary for tuples in the set R:

$$Q(R, \text{salary}, I) = \sum_{\bar{x} \in R \cap I} \text{salary}(\bar{x}) = f(28, 55K) + f(30, 58K) = 113K$$

An AVERAGE query for a measure function given a range is the ratio of the SUM query with the COUNT query for that range. Finally, taking $f(\bar{x}) \equiv \text{salary}(\bar{x}) \times \text{age}(\bar{x})$ the range-sum query computes the total product of salary and age for tuples in the set R:

$$Q(R, \text{salary} \times \text{age}, I) = \sum_{\bar{x} \in R \cap I} \text{salary}(\bar{x}) \times \text{age}(\bar{x}) = f(28, 55K) + f(30, 58K) = 3280M$$

This is an important component for the computation of covariance in the range R. In particular

$$\mathrm{Cov}(\text{age}, \text{salary}) = \frac{Q(R, \text{salary} \times \text{age}, I)}{Q(R, \mathbf{1}, I)} - \frac{Q(R, \text{age}, I) Q(R, \text{salary}, I)}{(Q(R, \mathbf{1}, I))^2}$$

The variance, kurtosis, or any other statistics in a range can be computed similarly.

Definition 2. *The* **data frequency distribution** *of* I *is the function* $\Delta_I : \mathrm{Dom}(F) \to \mathbb{Z}$ *that maps a point* $\bar{x}$ *to the number of times it occurs in* I*. To emphasize the fact that a query is an operator on the data frequency distribution, we write* $Q(R, f)|_I = Q(R, f, \Delta_I)$*. When* I *is clear from context we omit subscripts and simply write* Δ*.*

Example 2. For the table of Example 1, we have $\Delta(25, 50K) = \Delta(28, 55K) = \ldots = \Delta(57, 120K) = 1$ and $\Delta(\bar{x}) = 0$ otherwise.

Now we can rewrite the basic definition of the range-sum as.

$$Q(R, f, \Delta_I) = \sum_{\bar{x} \in \mathrm{Dom}(F)} f(\bar{x}) \chi_R(\bar{x}) \Delta_I(\bar{x}) \tag{1}$$

where χ_R is the characteristic function of the set R: $\chi_R(\bar{x}) = 1$ when $\bar{x} \in R$ and is zero otherwise. Equation 1 can be thought of as a vector dot product, where any function $g : \mathrm{Dom}(F) \to \mathbb{R}$ is a vector and for any $\bar{x} \in \mathrm{Dom}(F)$, $g(\bar{x})$ is the $\bar{x}$-coordinate of g. We denote this scalar product by $\langle g, h \rangle = \sum_{\bar{x} \in \mathrm{Dom}(F)} g(\bar{x})h(\bar{x})$. Allowing us to write

$$Q(R, f, \Delta_I) = \langle f\chi_R, \Delta_I \rangle \tag{2}$$

defining a range-sum query as a *vector query*: the scalar product of a function completely defined by the database instance (Δ_I) and another completely defined by the query ($f\chi_R$). We refer to the function $f\chi_R$ as the *query function*.

4 Wavelets

Before proceeding, we need some basic results about the fast wavelet transform. We refer the reader to [22] for a treatment similar to the one presented here. Readers familiar with Haar wavelets will see that the wavelet construction used in this paper is a generalization of the Haar transform which maintains many of the essential properties of the transform while providing more room for wavelet design.

We use wavelets arising from pairs of orthogonal convolution-decimation operators. One of these operators, H, computes a local average of an array at every other point to produce an array of *summary coefficients*. The other operator, G, measures how much values in the array vary inside each of the summarized blocks to compute an array of *detail coefficients*. Specifically we have filters h and g such that for an array $\mathbf{a}$ of length $2q$

$$H\mathbf{a}[i] = \sum_{j=0}^{2q-1} h[(2i-j) \mod 2q]\,\mathbf{a}[j] \text{ and } G\mathbf{a}[i] = \sum_{j=0}^{2q-1} g[(2i-j) \mod 2q]\,\mathbf{a}[j]$$

In order to ensure that H and G act as "summary" and "detail" filters, we also require that $\sum h[j] = \sqrt{2}$, $\sum g = 0$, $\sum h^2 = \sum g^2 = 1$, and $g[j] = (-1)^j h[1-j]$. These conditions imply that splitting an array into summaries and details preserves scalar products:

$$\sum_{j=0}^{2q-1} \mathbf{a}[j]\mathbf{b}[j] = \sum_{i=0}^{q-1} H\mathbf{a}[i]H\mathbf{b}[i] + \sum_{i=0}^{q-1} G\mathbf{a}[i]G\mathbf{b}[i] \tag{3}$$

Example 3. The Haar wavelet summary filter h is defined by $h[0] = h[1] = 1/\sqrt{2}$, and $h[i] = 0$ otherwise. The Haar wavelet detail filter has $g[0] = 1/\sqrt{2}$, $g[1] = -1/\sqrt{2}$, and $g[i] = 0$ otherwise. The convolution-decimation operators H and G corresponding to these filters are orthogonal.

To obtain the discrete wavelet transform, we continue this process recursively, at each step splitting the summary array from the previous step into a summary of summaries array and details of summaries array. In time $\Theta(N)$ we have computed the discrete wavelet transform.

Definition 3. *Given orthogonal convolution-decimation operators H and G, and an array* $\mathbf{a}$ *of length* 2^j*, the* discrete wavelet transform [DWT] *of* $\mathbf{a}$ *is the array* $\hat{\mathbf{a}}$ *where*

$$\hat{\mathbf{a}}[2^{j-j_*} + k] = GH^{j_*}\mathbf{a}[k]$$

for $1 \leq j_* \leq j$ *and* $0 \leq k < 2^{j-j_*}$*. Define* $\hat{\mathbf{a}}[0] = H^j\mathbf{a}[0]$*. We often refer to the elements of* $\hat{\mathbf{a}}$ *as the* wavelet coefficients *of* $\mathbf{a}$*. We refer to the arrays* $H^{j_*}\mathbf{a}$ *and* $GH^{j_*-1}\mathbf{a}$ *respectively as the* summary coefficients *and* detail coefficients *at level* j_**.*

The following consequence of Equation 3 is of fundamental importance

Lemma 1. *If* $\hat{\mathbf{a}}$ *is the DWT of* $\mathbf{a}$ *and* $\hat{\mathbf{b}}$ *is the DWT of* $\mathbf{b}$ *then* $\sum \hat{\mathbf{a}}[\eta]\hat{\mathbf{b}}[\eta] = \sum \mathbf{a}[i]\mathbf{b}[i]$*.*

To define wavelet transformations on multidimensional arrays we use the tensor product construction, which has appeared recently in the database literature as the foundation of Iterative Data Cubes [17]. The idea of this construction is simple: given a library of one dimensional transforms, we can build multidimensional transforms by applying one dimensional transforms in each dimension separately.

Specifically, we note that since the one dimensional DWT is linear, it is represented by a matrix $[W_{\eta,i}]$ such that for any array $\mathbf{a}$ of length N: $\hat{\mathbf{a}}[\eta] = \sum_{i=0}^{N-1} W_{\eta,i}\mathbf{A}[i]$ Given a multidimensional array $\mathbf{a}[i_0, \ldots, i_{d-1}]$, performing this transform in each dimension yields the *multivariate DWT* of $\mathbf{a}$

$$\hat{\mathbf{a}}[\eta_0, \ldots, \eta_{d-1}] = \sum_{i_0,\ldots,i_{d-1}=0}^{N-1} W_{\eta_0,i_0} W_{\eta_1,i_1} \cdots W_{\eta_{d-1},i_{d-1}} \mathbf{a}[i_0, \ldots, i_{d-1}]$$

Using the fast wavelet transform for each of these one dimensional matrix multiplications allows us to compute this sum in $\Theta(\ell N^d)$ for d-dimensional data. Repeated application of Lemma 1 yields

$$\sum_{\eta_0,\ldots,\eta_{d-1}=0}^{N-1} \hat{\mathbf{a}}[\eta_0, \ldots, \eta_{d-1}]\hat{\mathbf{b}}[\eta_0, \ldots, \eta_{d-1}] = \sum_{i_0,\ldots,i_{d-1}} \mathbf{a}[i_0, \ldots, i_{d-1}]\mathbf{b}[i_0, \ldots, i_{d-1}] \tag{4}$$

5 Naive Polynomial Range-Sum Evaluation Using Wavelets

Now we show how any range-sum query can be evaluated in the wavelet domain. This discussion allows us to see how data can be preprocessed using the wavelet transform, stored, and accessed for query evaluation. We also see that this method imposes no storage cost for dense data, and increases the storage requirements by a factor of $O(\log^d N)$ in the worst case. We fix orthogonal wavelet filters h and g of length ℓ.

Combining Equations 2 and 4, and interpreting functions on $\mathrm{Dom}(F)$ as d-dimensional arrays, we obtain a new formula for range-sums

$$Q(R, f, \Delta) = \langle \widehat{f\chi_R}, \hat{\Delta} \rangle = \sum_{\eta_0, \ldots, \eta_{d-1}=0}^{N-1} \widehat{f\chi_R}(\eta_0, \ldots, \eta_{d-1}) \hat{\Delta}(\eta_0, \ldots, \eta_{d-1}) \quad (5)$$

Giving a technique for evaluation of range-sums with arbitrary measures entirely in the wavelet domain: given a dataset with data frequency distribution Δ we preprocess it as follows

Wavelet Preprocessing

1. Prepare a [sparse] array representation of Δ. This requires time proportional to $|I|$, the size of the dataset.
2. Use the multidimensional fast wavelet transform to compute the [sparse] array representation of $\hat{\Delta}$. If the data are sparse, this requires time $O(|I|\ell^d \log^d N)$. If the data are dense, this requires time $N^d = O(|I|)$.
3. Store the array representation of $\hat{\Delta}$. If the data are sparse, use a hash-index. If the data are dense, use array-based storage. In either case we have essentially constant time access to any particular transform value.

For dense data, this preprocessing step introduces no storage overhead. The worst possible storage overhead arises when the dataset has only one record. $O(\ell^d \log^d N)$ nonzero wavelet transform values need to be stored in this case.

With our storage and access methods established, we now discuss query evaluation. In order to use Equation 5 to evaluate a general range-sum with query function $f\chi_R$ using the stored wavelet transform data, we proceed as follows

Naive Wavelet Query Evaluation

1. Compute the wavelet transformation $\widehat{f\chi_R}$. Using the fast wavelet transform requires time $O(\ell \log^d N)$. Initialize $\texttt{sum} \leftarrow 0$.
2. For each entry $\bar{\eta} = (\eta_0, \ldots, \eta_{d-1})$ in the array representation of $\widehat{f\chi_R}$, retrieve $\hat{\Delta}(\bar{\eta})$ from storage and set $\texttt{sum} \leftarrow \texttt{sum} + \widehat{f\chi_R}(\bar{\eta})\hat{\Delta}(\bar{\eta})$. For general query functions, there are $O(N^d)$ items retrieved from storage. When complete, sum is the query result.

This is a *correct* method for evaluating range-sum in the wavelet domain, but it is not an *efficient* method. In Section 6 we show that for polynomial range-sums it is possible to improve both the query transformation cost and the I/O cost dramatically.

6 Fast Range-Sum Evaluation Using Wavelets

With this background established, we present a first version of ProPolyne. For polynomial range-sums of degree less than δ in each attribute, this algorithm

has time complexity $O((2\ell \log N)^d)$ where $\ell = 2\delta + 1$. The data structure used for storage can be updated in time $O((\ell \log N)^d)$. To our knowledge there are no existing COUNT or SUM evaluation algorithms that provide faster query evaluation without having slower update cost. The algorithm requires preprocessing the data as described in Section 5. If the storage cost is prohibitive, it is possible to store a wavelet synopsis of the data and use this algorithm to evaluate approximate query results [21] or use this technique for dense clusters [11]. In Section 7 we show how ProPolyne can be refined to deliver good progressive estimates of the final query result by retrieving data in an optimal order.

At the heart of our query evaluation technique is a fast algorithm for computing wavelet transforms of query functions, and an observation that these transforms are very sparse. This allows us to evaluate queries using Equation 5 quickly.

6.1 Intuition

We can see why query functions can be transformed quickly and have sparse transforms by looking at a very simple example: consider the problem of transforming the indicator function χ_R of the interval $R = [5, 12]$ on the domain of integers from 0 to 15. We use the Haar filters defined in Example 3. Before computing the first recursive step of the wavelet transformation we already can say that

1. The detail coefficients are all zero unless the detail filter overlaps the boundary of R: $G\chi_R(i) = \chi_R(2i+1) - \chi_R(2i) = 0$ if $i \notin \{2, 6\}$
2. The summary coefficients are all zero except when the summary filter overlaps R: $H\chi_R(i) = 0$ when $i \notin [2, 6]$
3. The summary coefficients are constant when the summary filter is contained in R: $H\chi_R(i) = \sqrt{2}$ when $2 < i < 6$.

Without applying the operators H and G we are able to determine most of the values of $H\chi_R$ and $G\chi_R$ in constant time. The only interesting points, $2 = \lfloor \frac{5}{2} \rfloor$ and $6 = \lfloor \frac{12}{2} \rfloor$, arise from the boundaries of R. We can compute these values in constant time and see that $G\chi_R(2) = 1/\sqrt{2}, G\chi_R(6) = -1/\sqrt{2}$ and $H\chi_R(2) = H\chi_R(6) = 1/\sqrt{2}$. In constant time, we can build a data structure containing all of the information needed to evaluate any summary or detail coefficient. It turns out that we can always do this: at any recursive step in the Haar transform of a constant function the summary coefficients are constant on an interval, zero outside, and "interesting" on at most two boundary points. Also, there are at most two non-zero detail coefficients. Each of the $\log N = j$ steps can be carried out in constant time, allowing us to perform the entire transform in time and space $\Theta(\log N)$. Using the standard DWT algorithm would require time and space $\Theta(N)$. Because there are at most two nonzero detail coefficients per step, the resulting transform has less than $2 \log N$ nonzero terms. Carrying

our example through to the end we see that

$$\chi_R = \{0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0\}$$
$$\widehat{\chi_R} = \{2, \frac{1}{2}, \frac{3}{2\sqrt{2}}, -\frac{3}{2\sqrt{2}}, 0, \frac{1}{2}, 0, -\frac{1}{2}, 0, 0, \frac{1}{\sqrt{2}}, 0, 0, 0, -\frac{1}{\sqrt{2}}, 0\} \quad (6)$$

Now if we have stored the wavelet coefficients for a data density function Δ on $[0, 15]$ and want to use Equation 5 to evaluate $Q(R, f, \Delta)$ we only retrieve the values of $\hat{\Delta}$ where $\widehat{\chi_R} \neq 0$. Because there are at most $2 \log N$ of these coefficients, we can evaluate the query in time $O(\log N)$.

6.2 The Lazy Wavelet Transform

We now formalize these arguments and extend them to deal with general one dimensional polynomial range-sums. The important features of the Haar transformation noted above are (1) the recursive step of the DWT can be made in constant time and (2) the number of nonzero wavelet coefficients of the query function is $O(\log N)$. Now we show that with appropriate restrictions and choices of wavelet filters, we can obtain both of these features for general polynomial range-sums. We call this algorithm the *lazy wavelet transform* since it only evaluates convolution results when they are needed at the next recursive step. To make this work, we must use wavelets that satisfy a *moment condition*:

Definition 4. *G is said to satisfy condition $M(\delta)$ if $Gx^k = 0$ for $0 \leq k \leq \delta$. In other words, $\sum g[i]i^k = 0$ where g is the filter corresponding to the detail operator G.*

Example 4. The Db4 wavelet summary filter h_4 is defined by $h_4(0) = (1 + \sqrt{3})/(4\sqrt{2})$, $h_4(1) = (3 + \sqrt{3})/(4\sqrt{2})$, $h_4(2) = (3 - \sqrt{3})/(4\sqrt{2})$, $h_4(3) = (1 - \sqrt{3})/(4\sqrt{2})$, and $h_4(i) = 0$ otherwise. The Db4 wavelet detail filter has $g_4(i) = (-1)^i h_4(1-i)$. The convolution-decimation operators corresponding to this pair of filters are orthogonal and satisfy $M(1)$. The Haar filters satisfy $M(0)$. For higher moment conditions, we can use higher order Daubechies filters [22, 14, 4].

Now we can state the result that lies at the core of our technique.

Theorem 1 (Lazy Wavelet Transform). *Assume that $f\chi_R$ is a query function for a polynomial range-sum with $\deg f = \delta$ on a domain of size N, and that h and g are filters of length ℓ generating orthogonal convolution-decimation operators that satisfy $M(\delta)$. Then the sparse representation of the wavelet transform of $f\chi_R$ with filters h and g can be computed in time $\Theta(\ell \log N)$ and the resulting transform has at most $2(\ell - 1) \log N$ nonzero values.*

To prove this result we need two lemmas which formally state our intuitive notion that most of the wavelet coefficients for a query function do not need to be computed. In the statement of these lemmas, f is a polynomial of degree less than or equal to δ, and the orthogonal wavelet filters h and g with filter length ℓ satisfy $M(\delta)$. Recall that we take $N = 2^j$ throughout this paper.

Lemma 2. *For the range function $f\chi_{[l,\,u]}$ on $[0, 2^j - 1]$, the summary coefficients $S(x, j_*)$ at level $j_* \leq 0$ are zero outside an interval $[l^s_{j_*} - \ell, u^s_{j_*} + \ell]$ and are equal to a polynomial $p_{j_*}(x)$ of degree less than or equal to δ in the interval $[l^s_{j_*}, u^s_{j_*}]$. The values $l^s_{j_*}, u^s_{j_*}$, and the coefficients of p_{j_*} can be computed in time $O(\delta\ell)$ from the values $l^s_{j_*+1}, u^s_{j_*+1}$, and the coefficients of p_{j_*+1}.*

Proof: This is trivial for $j_* = 0$, where the summary coefficients are just the original range function. We can take $l^s_0 = l$ and $u^s_0 = h$. Inductively assume that the result holds for $j_* + 1 \leq 0$.

By definition

$$S(x, j_*) = \sum_{k=0}^{\ell-1} h_k S(2x - k, j_* + 1)$$

If $2x < l^s_{j_*+1} - \ell$ or if $2x > u^s_{j_*+1} + 2\ell - 1$ then the induction hypothesis implies that $S(x, j_*)$ is zero. Also, if $l^s_{j_*+1} + \ell - 1 \leq 2x \leq u^s_{j_*+1}$ then

$$S(x, j_*) = \sum_{k=0}^{\ell-1} h_k p_{j_*+1}(2x - k) \equiv p_{j_*+1}(x)$$

Where we can compute the coefficients of $p_{j_*+1}(x) \equiv \sum_{k=0}^{\ell-1} h_k p_{j_*+1}(2x - k)$ in time $O(\delta\ell)$. Choosing $l^s_{j_*} = \lceil \frac{l^s_{j_*+1}+\ell-1}{2} \rceil$ and $u^s_{j_*} = \lfloor \frac{u^s_{j_*+1}}{2} \rfloor$ we find that the lemma holds for level j_*. Notice that it may be the case that $l^s_{j_*} \geq u^s_{j_*}$, giving us effectively one short interval of nonzero coefficients. This completes the induction step and the proof is complete. □

Given this simple structure of the summary coefficients it is now very easy to say without computation that most of the detail coefficients are zero, and that the location of the few nonzero terms is easy to compute.

Lemma 3. *The detail coefficients of the range function in Lemma 2 at level $j_* < 0$ are supported in two (not necessarily disjoint) intervals $[l^d_{j_*}, l^d_{j_*} + \ell]$ and $u^d_{j_*}, u^d_{j_*} + \ell]$ where $l^d_{j_*} = \lfloor \frac{l^s_{j_*+1}-2\ell+2}{2} \rfloor$ and $u^d_{j_*} = \lceil \frac{u^s_{j_*+1}-\ell+2}{2} \rceil$.*

Proof: This is very similar to the proof of Lemma 2, only the moment condition for the filter G now makes the terms between $l^d_j + \ell$ and u^d_j vanish. The fact that we chose G to be supported on the interval $[2 - \ell, 1]$ also affects the bookkeeping that leads to the final form of the results. □

Proof of Theorem 1: With the lemmas above we see that each of the $\log N$ recursive steps of the wavelet transform can be carried out in time $O(\ell)$ because there are only $O(\ell)$ "interesting" coefficients to compute. This gives a total time complexity of $O(\ell \log N)$. Lemma 3 implies that at each of the $\log N$ resolution levels at most $2\ell - 2$ nonzero detail coefficients are added, completing the proof. □

Note that we can use Daubechies' construction of compactly supported orthogonal wavelets to produce wavelets satisfying $M(\delta)$ that have filter length $\ell = 2\delta + 2$. Using these filters it is possible to transform polynomial query functions of degree less than δ in time $\Theta(\delta \log N)$ and the result has less than $(4\delta + 2) \log N$ nonzero coefficients.

6.3 Polynomial Range-Sum Evaluation

In Section 5 we discussed how wavelet data can be preprocessed, stored, and accessed. Now we show how using the lazy wavelet transform can dramatically speed up query evaluation in the wavelet domain. First we define a special type of polynomial range query that is particularly easy to work with.

Definition 5. *A polynomial range-sum with measure function* $f : \mathrm{Dom}(F) \to \mathbb{R}$ *is said to satisfy condition* $S(\delta)$ *if* $f(x_0, \ldots, x_{d-1}) = \prod_{i=0}^{d-1} p_i(x_i)$ *and each* p_i *has degree less than or equal to* δ*, and* R *is a hyper-rectangle.*

All of the queries in Example 1 satisfy this condition: COUNT satisfies $S(0)$, SUM, AVERAGE, and COVARIANCE are computed with range-sums satisfying $S(1)$, and VARIANCE is computed with range-sums satisfying $S(2)$. As Example 1 makes clear, the class of polynomial range-sums is rich, even restricted to satisfy condition $S(\delta)$.

Consider a polynomial range-sum with query function $f\chi_R = \prod_{j=0}^{d-1} p_j(i_j)\chi_{R_j}(i_j)$. By Equation 4 $\widehat{f\chi_R} = \prod \widehat{p_j\chi_{R_j}} = \prod \widehat{p_j\chi_{R_j}}$. If $f\chi_R$ satisfies $S(\delta)$, then using Daubechies' filters of length $2\delta + 2$, we can compute (a data structure representing) $\widehat{f\chi_R}$ in time $\Theta(d\delta \log N)$, and $\widehat{f\chi_R}$ has $O((4\delta + 2)^d \log^d N)$ nonzero coefficients. Thus we have the following query evaluation technique, which we present for queries satisfying $S(\delta)$. General multivariate polynomials of degree δ can always be split into a sum of polynomials satisfying $S(\delta)$, each of which is transformed separately.

Fast Wavelet Query Evaluation

1. Use the lazy wavelet transform to compute the d one-dimensional wavelet transforms $\widehat{p_j\chi_{R_j}}$. Initialize $\texttt{sum} \leftarrow 0$.
2. Iterate over the Cartesian product of the nonzero coefficients of the $\widehat{p_j\chi_{R_j}}$. For each $\bar{\eta} = (\eta_0, \ldots, \eta_{d-1})$ in this set, retrieve $\hat{\Delta}(\bar{\eta})$ from storage and set $\texttt{sum} \leftarrow \texttt{sum} + \hat{\Delta}(\bar{\eta}) \prod \widehat{p_j\chi_{R_j}}(\eta_j)$. When complete, sum is the query result.

Notice that this algorithm is dominated by the I/O phase (step 2) for $d > 1$. The online preprocessing phase (step 1) takes time and space $\Theta(d\delta \log N)$.

The lazy wavelet transform also gives us a way to perform fast updates of the database. To insert a record $\bar{i} = (i_0, \ldots, i_{d-1})$, let $\chi_{\bar{i}}$ denote the function equal to 1 at $\bar{i}$ and zero elsewhere. Then the updated data frequency distribution is $\Delta_{\text{new}} = \Delta + \chi_{\bar{i}}$. By the linearity of the wavelet transform, we have $\widehat{\Delta_{\text{new}}} = \hat{\Delta} + \hat{\chi}_{\bar{i}}$. Thus to perform an update, we simply compute $\hat{\chi}_{\bar{i}}$ and add the results to storage. Furthermore, $\chi_{\bar{i}}$ can be thought of as a query function satisfying $S(0)$, so we can transform it just as we transform other query functions. A careful look reveals that in this case the "interesting intervals" overlap completely, and $\hat{\chi}_{\bar{i}}$ can be computed in time $\Theta((2\delta + 1)^d \log^d N)$.

Wavelet Update

1. To insert $\bar{i} = (i_0, \ldots, i_{d-1})$ use the lazy wavelet transform to compute $\hat{\chi_{i_j}}$ for $0 \leq j < d$.
2. For each $\bar{\eta} = (\eta_0, \ldots, \eta_{d-1})$ in the Cartesian product of the nonzero entries of $\hat{\chi_{i_j}}$, set $\hat{\Delta}(\bar{\eta}) \leftarrow \hat{\Delta}(\bar{\eta}) \prod \hat{\chi_{i_j}}(\eta_j)$. For hash table access, this may require an insertion.

Theorem 2. *Using Daubechies' wavelets with filter length $2\delta+2$, the* Wavelet Preprocessing *strategy to store data,* Wavelet Update *to insert new records, and* Fast Wavelet Query Evaluation *to evaluate queries it is possible to evaluate any polynomial range-sum satisfying $S(\delta)$ in time $O((4\delta+2)^d \log^d N)$. It is possible to insert one record in time $O((2\delta+1)^d \log^d N)$.*

7 Progressive Polynomial Range-Sum Evaluation

Note that in Equation 6 some of these coefficients are larger than others. For larger ranges and higher dimensions, this phenomenon becomes dramatic: most of the information is contained in a handful of coefficients. Intuitively, if we use the largest coefficients first we should obtain an accurate estimate of the query result before the computation is complete. This is the basic idea behind ProPolyne, but before proceeding we pause to ask precisely why we think that evaluating queries using the 'big' coefficients first gives better approximate answers. Once we can state precisely how this evaluation order is better, we prove that ProPolyne uses the best possible evaluation order.

Definition 6. *A* progressive evaluation plan *for a sum $S = \sum_{0 \leq i < N} \mathbf{a}[i]$ is a permutation σ of the integers 0 to $N-1$. The estimate of S at the j*th progressive step *of this plan is $\sum_{0 \leq i < j} \mathbf{a}[\sigma(i)]$.*

7.1 Query Best-B ProPolyne

How do we determine whether one progressive evaluation plan produces better results than another? When evaluating range aggregate queries using Equation 5, we must choose a data independent progressive evaluation plan for a sum of the form $\sum \widehat{f\chi_R}(i)\hat{\Delta}(i)$. We cannot look at $\hat{\Delta}$ until executing our plan, but we want to minimize the average square error observed at each progressive step when operating on a random dataset.

Definition 7. *Let $\tilde{Q}_1(\Delta)$ and $\tilde{Q}_2(\Delta)$ be approximations of the query $Q(R, f, \Delta)$. We say $\tilde{Q}_1$* dominates *$\tilde{Q}_2$ if*

$$E_\Delta[(\tilde{Q}_1(\Delta) - Q(R, f, \Delta))^2] \leq E_\Delta[(\tilde{Q}_2(\Delta) - Q(R, f, \Delta))^2]$$

where Δ is randomly selected from the set $\{\Delta \mid \sum \Delta^2(i) = 1\}$. We say that one progressive query plan dominates another if the estimate of the first plan dominates the estimate of the second at every progressive step.

We want to find a progressive evaluation plan that dominates all others. It is not obvious that this is possible, but the following result shows that the 'biggest coefficient' plan suggested at the beginning of the section is in fact the best.

Theorem 3. *Let y be a vector randomly selected from the set $\{y \mid \sum_{i=0}^{N-1} y_i^2 = 1\} = S^{N-1}$ with uniform distribution, let $I \subset [0, N-1]$ be a set of size B, and for a vector x let $\tilde{x}_I$ denote x with all coordinates except those in I set to zero. Denote the set of the B biggest (largest magnitude) coordinates of x by I^*. Then for any choice of I we have*

$$E_y[\langle x - \tilde{x}_{I^*}, y\rangle^2] \leq E_y[\langle x - \tilde{x}_I, y\rangle^2]$$

In other words, approximating x with its biggest B *terms gives us the* best B *term approximation of $\langle x, y\rangle$- an approximation that dominates all others.*

Proof: For any I, y we have $\langle x - \tilde{x}_I, y\rangle = \sum_{i,j \notin I} y_i(x_i x_j) y_j = y^T R y$, where $R = \tilde{x}_I^T \tilde{x}_I$ is a symmetric matrix. The average error can be obtained by integrating over the sphere of all y's

$$E_y[\langle x - \tilde{x}_I, y\rangle^2] = \int_{S^{N-1}} y^T R y = \int_{S^{N-1}} y^T D y = \sum \lambda_i \int_{S^{N-1}} y_i^2 = \frac{1}{n} \operatorname{trace}(R)$$

where D is the diagonalization of R and λ_i are the eigenvalues. This chain of equalities is possible because R is symmetric, hence is diagonalized by a unitary transformation that preserves the uniform distribution. But the trace of R is just the sum of squares of the coordinates of $\tilde{x}_I$, so $E_y[\langle x - \tilde{x}_I, y\rangle^2] = \frac{1}{n} \sum_{i \notin I} x_i^2$, which is clearly minimized by taking $I = I^*$. □

Corollary 1 *The progressive query plan obtained by using the largest query coefficients first dominates all other plans.*

This evaluation plan is the foundation for our progressive algorithm, ProPolyne. Because after B progressive steps ProPolyne provides the best-B wavelet approximation of a query, we refer to this technique as *query best-B ProPolyne*. We implement this progressive query plan by first evaluating the query function transformation using Theorem 1, then building a heap from the resulting set of nonzero wavelet coefficients. Compute the sum repeatedly extracting the top element from this heap- the partial sums provide accurate progressive query estimates. As described in Section 8, this analytical framework also provides efficiently computable guaranteed error bounds at each progressive step.

7.2 Data best-B ProPolyne

Previous uses of wavelets for approximate query evaluation have focused on *data approximation*, using wavelets to produce a precomputed synopsis data structure. The reader may note that the rôle of the data and the query in the results of Section 7.1 were entirely symmetric. If we can say "the biggest-B

approximation of a query is the best-B approximation for random data", we can just as easily say "the biggest-B approximation of the data is the best-B approximation for random queries". Hence we can obtain a different sort of progressive query evaluation by sorting the data wavelet coefficients offline, then retrieving them in decreasing order of magnitude. This is the spirit of the approximate query evaluation algorithms of [21], where it is shown that this gives reasonable estimates quickly. The technique presented here has the extra benefit of treating measure dimensions symmetrically, supporting general polynomial range-sums. We call this technique *data best-B ProPolyne*.

Unfortunately, query workloads are very far from being randomly distributed on the unit sphere. In practice, the best ordering would be weighted by the expected query workload. In any case, this technique only works well if the data are well approximated by the chosen wavelets. This stands in contrast to query best-B ProPolyne, where the query functions are always well approximated by wavelets. In Section 9 we see that the performance of data best-B ProPolyne varies dramatically with the dataset, and is consistently less accurate than query best-B ProPolyne.

7.3 Evaluation of Fixed-Measure SUM Queries

In practice, there are situations where the measures and aggregate functions are known at the time the database is built. When this is the case, ProPolyne can be optimized. In particular, it can be adapted to operate on the wavelet transform of the measure function rather than the frequency distribution. We call this adaptation *fixed measure* ProPolyne, or ProPolyne-FM. One notable optimization that ProPolyne-FM allows is the use of Haar wavelets in all dimensions. The fact that one-dimensional Haar wavelets do not overlap brings query evaluation cost down to $(2j)^{d-1}$ and update cost to j^{d-1} for a table with $d-1$ j-bit attribute and one measure attribute. We note that this cost is identical to that of the Space-efficient Dynamic Data Cube [16]. ProPolyne-FM serves another useful rôle : it solves the same problem as other pre-aggregation methods, so it is directly comparable.

The process of turning ProPolyne-FM into a data or query best-B progressive algorithm is identical to the process for unrestricted ProPolyne. It happens that when using Haar wavelets, the data best-B version of ProPolyne-FM is simply a progressive implementation of the *compact data cube* presented in [21]. The query best-B version of ProPolyne-FM is novel, and we see in Section 9 that approximate results it produces are significantly more accurate than those produced by the compact data cube. For the remainder of the paper we use ProPolyne-FM to mean this query best-B version.

8 Guaranteed Error Bounds

An important component of any approximation technique is its ability to provide meaningful information about the distribution of errors. In this section we

present a family of analytic absolute bounds for the error of estimates made using the algorithms of Section 7. These bounds can be maintained efficiently and made available throughout progressive query evaluation.

PROPOLYNE evaluates queries by selecting the most important wavelets for the query and evaluating Equation 5 using these terms first. The error resulting from using only the best B coefficients is straightforward, if not efficient, to compute: simply compute the scalar product using the least important $2^j - B$ coefficients. We produce a tractable bound on this value by using Hölder's inequality in each resolution level. Specifically, if Ξ_B denotes the set of indices of the most important B wavelets, $\tilde{Q}_B(R, f, \Delta)$ is the approximate range sum obtained using only wavelets in Ξ_B, and Λ denotes the set of wavelet resolution levels, then we have the following error bound:

$$|Q(R, f, \Delta) - \tilde{Q}_N(R, f, \Delta)| \\ \leq \sum_{\lambda \in \Lambda} \left[\sum_{\bar{\eta} \in \lambda \setminus \Xi_B} |\widehat{f\chi_R}(\bar{\eta})|^{p_\lambda} \right]^{1/p_\lambda} \left[\sum_{\bar{\eta} \in \lambda \setminus \Xi_B} |\hat{\Delta}(\bar{\eta})|^{p_\lambda/(p_\lambda - 1)} \right]^{(p_\lambda - 1)/p_\lambda} \quad (7)$$

where $1 \leq p_\lambda \leq \infty$ for each λ. It is possible to achieve most of the benefit of the bound in (7) more cheaply by identifying a small set of resolution levels that contain most of the energy for the data density distribution or for an expected query workload. All other resolution levels are grouped into one large level and estimate (7) is maintained for the reduced set.

9 Experimental Results

In this section we present results from our experiments with ProPolyne and related algorithms in order to provide the reader with an overview of how these techniques perform on real-world data. ProPolyne-FM's worst-case performance as an exact algorithm (which is very similar to the performance of ProPolyne) is compared with related exact pre-aggregation methods in Table 1. Our focus in this section is on the accuracy of ProPolyne's progressive estimates. Our experiments show that wavelet-based query approximation delivers consistently accurate results, even on datasets that are poorly approximated by wavelets. Not only is the performance consistent, it is consistently better than data approximation. By directly comparing our methods with the wavelet-based data compression method proposed by Vitter and Wang [21] we see that query approximation based ProPolyne delivers significantly more accurate results after retrieving the same number of values from persistent storage.

9.1 Experimental Setup

We report results from experiments on three datasets. PETROL is a set of petroleum sales volume data with 56504 tuples, sparseness of 0.16%, and five dimensions: location, product, year, month, and volume (thousands of gallons). PETROL is our example of a dataset for which traditional data approximation works well [1].

Algorithm	Query Cost	Update Cost	Storage Cost
ProPolyne-FM	$(2\log N)^d$	$\log^d N$	$\min\{\lvert I\rvert \log^d N, N^d\}$
SDDC [16]	$(2\log N)^d$	$\log^d N$	N^d
Prefix-Sum [11]	2^d	N^d	N^d
Relative Prefix-Sum [6]	2^{2d}	$N^{d/2+1}$	N^d

Table 1. Query/update/storage tradeoff for several exact SUM algorithms

GPS[1] is a set of sensor readings from a group of GPS ground stations located throughout California. We use a projection of the available data to produce a dataset with 3358 tuples, sparseness of 0.01%, and four dimensions: latitude, longitude, time, and height velocity. The presence of a tuple (lat, long, t, v) means that a sensor observed that the ground at coordinates (lat, long) was moving upward with a velocity of v at time t. GPS is our example of a dataset for which traditional data approximation works poorly.

TEMPERATURE is a dataset holding the temperatures at points all over the globe and at 20 different altitudes on March 1, 2001. It has 207360 tuples, sparseness of 1.24%, and four dimensions: latitude, longitude, altitude, and temperature. The TEMPERATURE dataset is considerably larger than the GPS and PETROL datasets, and we use it to emphasize the fact that as datasets get larger, the benefit of using ProPolyne increases.

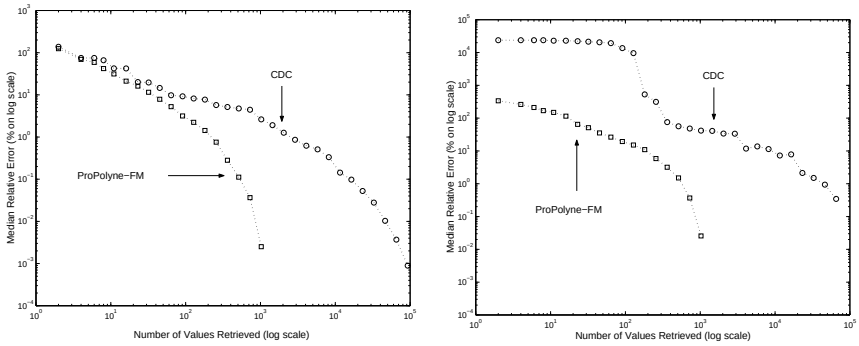
For all tests, 250 range queries were generated randomly from the set of all possible ranges with the uniform distribution. If a generated range selects fewer than 100 tuples from the dataset, it is discarded and a new range is generated.

All graphs display the progressive accuracy improvement of various approximation techniques for queries on a single dataset. The horizontal axis always displays the number of values retrieved by an algorithm on a logarithmic scale. The vertical axis of each graph displays the median relative error for a set of generated queries. Relative error is used so that queries returning large results do not dominate the statistics. Median error is used rather than mean error in order to avoid the noise caused by the one-sided fat tail of observed relative error. The results using mean error are qualitatively similar, but are not as smooth.

9.2 Performance for Fixed Measure Range-sums

Figure 1 compares the performance of ProPolyne-FM with a progressive version of the *compact data cube* (CDC) [21] on the PETROL and GPS datasets. Other techniques, including [9], have been compared favorably to the CDC. We do not directly compare ProPolyne to these because they have no progressive analog. We see that CDC works very well on the PETROL dataset, producing a median relative error under 10% after using less than 100 wavelet coefficients. Still, ProPolyne-FM works better than CDC from the beginning, and this difference only grows as evaluation progresses. The difference between the performance of

[1] We would like to thank our colleagues at JPL, Brian Wilson and George Hajj, for providing us with the GPS and TEMPERATURE datasets.



(a) PETROL (mean selectivity: 22.3%)

(b) GPS (mean selectivity: 20.4%)

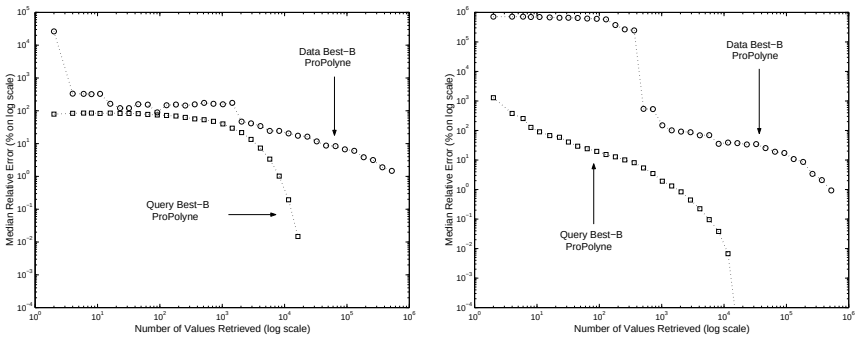
Fig. 1. Progressive accuracy for Compact Data Cube (CDC) [21] and ProPolyne-FM.

the two techniques on the GPS dataset is striking: CDC must use more than five times as many wavelet coefficients as there were tuples in the original table before providing a median relative error of 10%. ProPolyne-FM reaches this level of accuracy after retrieving less than 300 wavelet coefficients.

9.3 Performance for General Range-sums

Figure 2 compares the performance of data best-B ProPolyne and query best-B ProPolyne on the PETROL and GPS datasets. Unlike the previous section, the queries for these tests slice in all dimensions, including the measure dimension. Data best-B ProPolyne can be thought of as an extension of CDC that supports this richer query set. As in the previous section, the method based on query approximation consistently and significantly outperforms the analogous data compression method. By the time query best-B ProPolyne has achieved a median error of 10%, data best-B ProPolyne still has a median error of near 100% for the PETROL dataset. The data best-B ProPolyne error for the GPS dataset at this point is enormous. Notice also that the data best-B results exhibit accuracy "cliffs" where the progression reaches a set of coefficients that are particularly important for the given query workload. This hints that query workload information is critical to improving the ordering of data best-B coefficients.

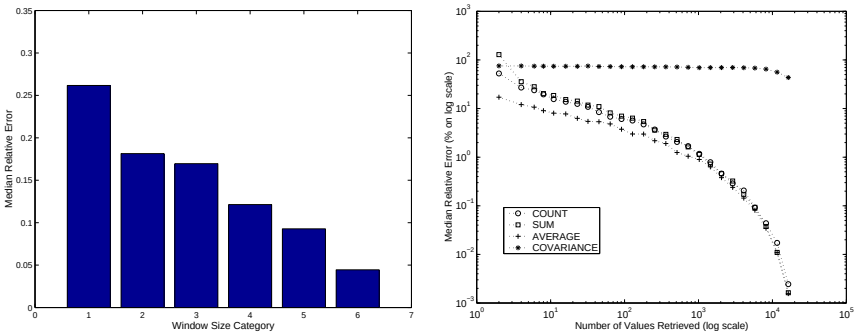
Finally, Figure 3 illustrates the progressive accuracy of query best-B ProPolyne On the TEMPERATURE dataset. Figure 3(a) displays relative error for AVERAGE queries on randomly generated ranges of different sizes. We define the size of a range to be the product of the lengths of its sides. Larger ranges have better approximate results, suggesting that a basis other than wavelets may provide better approximation of query workloads with small ranges.



(a) PETROL (mean selectivity: 18.2%)

(b) GPS (mean selectivity: 17.5%)

Fig. 2. Progressive query accuracy for data best-B and query best-B ProPolyne.



(a) Relative error vs. range size after retrieving 500 values

(b) Second order statistics (mean selectivity 16.5%)

Fig. 3. Query Best-B ProPolyne on the TEMPERATURE dataset. For (a), range size categories are as follows. category 1: size $<$ 5000, category 2: 5000 $\leq$ size $<$ 10000, category 3: 10000 $\leq$ size $<$ 20000, category 4: 20000 $\leq$ size $<$ 40000, category 5: 40000 $\leq$ size $<$ 80000, category 6: 80000 $\leq$ size.

Figure 3(b) displays progressive relative error for COUNT, SUM, AVERAGE, and COVARIANCE queries on the TEMPERATURE dataset. Here we note that COUNT, SUM, and AVERAGE all obtain excellent accuracy after retrieving a very small number of wavelet coefficients. AVERAGE is significantly more accurate early in the computation, obtaining a median relative error below 10% using just 16 data wavelet coefficients. COUNT and SUM both achieve this level of accuracy using close to 100 data wavelet coefficients. COVARIANCE stands out by not having significantly improving accuracy until near the end of the computation. We emphasize that we obtain exact results for COVARIANCE just as quickly as for other query types. This slow convergence largely due to the fact that we compute the covariance by subtracting two large approximate numbers to obtain a relatively small number.

10 Conclusions and Future Plans

In this paper we present ProPolyne, a novel MOLAP pre-aggregation strategy which can be used to support conventional queries such as COUNT and SUM alongside more complicated *polynomial range-sums*. ProPolyne is the first pre-aggregation strategy that does not require measures to be specified at the time of database population. Instead, measures are treated as functions of the attributes which can be specified at query time. This approach leads naturally to a new *data independent* progressive and approximate query answering technique which delivers excellent results when compared to other proposed data compression methods. ProPolyne delivers all of these features with provably poly-logarithmic worst-case query and update cost, and with storage cost comparable to or better than other pre-aggregation methods.

We intend to extend this work in several ways. Preliminary experiments indicate that using synopsis information about query workloads or data distributions can dramatically improve sort orders for both query best-B and data best-B techniques. Dimensionality reduction techniques can improve I/O complexity at the expense of some accuracy in the final results. As presented here, ProPolyne requires random access to stored data; we will explore clustering strategies which take advantage of ProPolyne's unique access patterns. Finally we wish to explore the limits of linear algebraic query approximation for approximate query answering. This includes finding complexity lower bounds, investigating more complex queries (e.g. OLAP drill-down, relational algebra), and making an efficient adaptive choice of the best basis for evaluating incoming queries.

References

1. J. L. Ambite, C. Shahabi, R. R. Schmidt, and A. Philpot. Fast approximate evaluation of OLAP queries for integrated statistical data. In *Nat'l Conf. for Digital Government Research, Los Angeles*, May 2001.
2. K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Proc. VLDB*, pages 111–122, 2000.

3. C.-Y. Chan and Y. E. Ionnidis. Hierarchical cubes for range-sum queries. In *Proc. VLDB*, pages 675–686, 1999.
4. I. Daubechies. Orthonormal bases of compactly supported wavelets. *Comm. Pure and Appl. Math.*, 41:909–996, 1988.
5. S. Geffner, D. Agrawal, and A. E. Abbadi. The dynamic data cube. In *Proc. EDBT*, pages 237–253, 2000.
6. S. Geffner, D. Agrawal, A. E. Abbadi, and T. Smith. Relative prefix sums: An efficient approach for querying dynamic OLAP data cubes. In *Proc. ICDE*, pages 328–335, 1999.
7. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Optimal and approximate computation of summary statistics for range aggregates. In *Proc. ACM PODS*, pages 228–237, 2001.
8. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. VLDB*, 2001.
9. D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *Proc. ACM SIGMOD*, pages 463–474, 2000.
10. J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *Proc. ACM SIGMOD*, pages 171–182, 1997.
11. C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *Proc. ACM SIGMOD*, pages 73–88, 1997.
12. I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *Proc. ACM SIGMOD*, pages 401–412, 2001.
13. V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *Proc. SSDBM*, pages 24–33, 1999.
14. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* Cambridge Univ. Press, 1992.
15. M. Riedewald, D. Agrawal, and A. E. Abbadi. pCube: Update-efficient online aggregation with progressive feedback. In *Proc. SSDBM*, pages 95–108, 2000.
16. M. Riedewald, D. Agrawal, and A. E. Abbadi. Space-efficient datacubes for dynamic environments. In *Proc. of Conf. on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 24–33, 2000.
17. M. Riedewald, D. Agrawal, and A. E. Abbadi. Flexible data cubes for online aggregation. In *Proc. ICDT*, pages 159–173, 2001.
18. R. R. Schmidt and C. Shahabi. ProPolyne: A fast wavelet-based technique for fast evaluation of polynomial range-sum queries. In *Proc. EDBT*, Springer, 2002.
19. R. R. Schmidt and C. Shahabi. Wavelet based density estimators for modeling OLAP data sets. In *SIAM Workshop on Mining Scientific Datasets*, Chicago, April 2001. Available at http://infolab.usc.edu/publication.html.
20. J. Shanmugasundaram, U. Fayyad, and P. Bradley. Compressed data cubes for OLAP aggregate query approximation on continuous dimensions. In *Proc. SIGKDD*, August 1999.
21. S.-C. Shao. Multivariate and multidimensional OLAP. In *Proc. EDBT*, pages 120–134, 1998.
22. J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proc. ACM SIGMOD*, pages 193–204, 1999.
23. M. V. Wickerhauser. *Adapted Wavelet Analysis: From Theory to Software.* IEEE Press, 1994.

24. Y.-L. Wu, D. Agrawal, and A. E. Abbadi. Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes. In *Proc. CIKM*, pages 414–421, 2000.