How to Evaluate Multiple Range-Sum Queries Progressively*

Rolfe R. Schmidt
University of Southern California
Computer Science Department
Los Angeles, California
rrs@usc.edu

Cyrus Shahabi
University of Southern California
Computer Science Department
Los Angeles, California
shahabi@usc.edu

ABSTRACT

Users of decision support system typically submit batches of range-sum queries simultaneously rather than issuing individual, unrelated queries. We propose a wavelet based technique that exploits I/O sharing across a query batch to evaluate the set of queries progressively and efficiently. The challenge is that now controlling the structure of errors across query results becomes more critical than minimizing error per individual query. Consequently, we define a class of structural error penalty functions and show how they are controlled by our technique. Experiments demonstrate that our technique is efficient as an exact algorithm, and the progressive estimates are accurate, even after less than one I/O per query.

1. INTRODUCTION

Range aggregate queries can be used to provide database users a high level view of massive or complex datasets. These queries have been widely studied, but much of the research has focused on the evaluation of individual range queries. However, in many applications, whether the users are humans or data mining algorithms, range aggregate queries are issued in structured batches.

For example, data consumers often specify a coarse partition of the domain of a data set, and request aggregate query results from each cell of this partition. This provides a data synopsis which is used to identify interesting regions of the data domain. Users then drill-down into the interesting regions by partitioning them further and requesting aggregate query results on the new cells. The data consumer is interested in both the individual cell values that are returned and the way the query results change between neighboring cells.

It is possible to evaluate a batch of range aggregate queries by repeatedly applying any exact, approximate, or progressive technique designed to evaluate individual queries. While flexible, this approach has two major drawbacks:

1. I/O and computational overhead are not shared between queries.
2. Approximate techniques designed to minimize single-query error cannot control *structural error* in the result set. For example, it is impossible to minimize the error of the difference between neighboring cell values.

Both of these issues require treating the batch as a single query, not as a set of unrelated queries. Item 2 is particularly important. An OLAP system user searching for "interesting" areas in a dataset might be more concerned with finding large cell to cell changes in a measure than accurately identifying the actual cell values. In this situation the *structure* of the errors is more important than the total size of the errors. The particular structural error of interest may change from query to query: some queries may need to minimize the sum of square errors, others may be interested in finding "temporal surprises", and others may be searching for local maxima. If only a subset of results are currently being rendered on the screen, these should be prioritized. Ideally, the *structural error penalty* function could be part of a query submitted to an approximate query answering system. The system then produces its best approximate result for that penalty function.

1.1 Related Work

Extensive research has been done developing techniques for exact [2, 3, 8, 12, 14], approximate [4, 6, 10, 15], and progressive [9, 11, 18] evaluation of *single* range-sum queries. While experimental results suggest that these techniques are very effective, they do not address resource sharing between queries or control of structural error in query batches. Multi-query optimization in OLAP has been addressed [19]. The primary focus is on sharing of resources and scans, and selection of precomputed group-bys. These exact methods produce considerable savings, but are based on the assumption that some relation must be scanned in order to answer

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and IIS-0082826, NIH-NLM grant nr. R01-LM07061, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash/equipment gifts from Okawa Foundation, Microsoft, NCR and SUN.

the batch of queries. This will not be the case when relations are stored using pre-aggregation techniques such as prefix-sums [8], wavelets [14], or general Iterative Data Cubes [12].

Proposed techniques for online aggregation [7] provide a way to simultaneously answer a batch of queries using progressive approximations of the underlying dataset. Priority can be placed on different cells, giving users control over a form of structural error. For appropriate data this technique provides accurate answers quickly, but the entire relation must be viewed before results become exact.

Wavelets are often used for data compression, and have been proposed as a tool for producing pre-computed synopses of datasets to support approximate query answering [17]. Wavelet-based precomputed synopses for relational algebra [1] provide approximate batch query results, but give no control over structural error in ad hoc queries. As with any data compression based technique, this is only effective when the data are well approximated by wavelets. This paper takes a fundamentally different approach, and uses wavelets to approximate queries, not data.

1.2 Contributions

This paper explores how query approximation can be used as an alternative to data approximation to provide efficient progressive query answering tuned to an arbitrary penalty function. The major contributions of this paper are

- Batch-Biggest-B, an exact wavelet-based query evaluation strategy that exploits I/O sharing to provide fast exact query results using a data structure that can be updated efficiently.
- The introduction of structural error for batch queries, and the definition of structural error penalty functions. This generalizes common error measures such as sum of square errors (SSE) and L^p norms.
- A progressive version of Batch-Biggest-B that can accept any penalty function specified at query time, and will minimize the worst case and average penalty at each step of the computation.

Finally, we note that nowhere do our methods rely on the fact that we use the wavelet transform to preprocess the data. We can use any linear transformation of the data that has a left inverse as a *storage strategy*. We can use the left inverse to rewrite query vectors to their representation in the transformation domain, giving us an *evaluation strategy*. Examples of linear storage/evaluation strategies include full query precomputation, no precomputation, and prefix-sum precomputation [8]. In fact, any Iterative Data Cube [12] is a linear storage/evaluation strategy. Batch-Biggest-B can be used with any structural error penalty to turn any of these single-query exact evaluation strategies into an I/O efficient progressive evaluation method for batch queries.

Why use wavelets then? Wavelets provide an update efficient storage format for which range-queries can be answered quickly. Furthermore, as pointed out in [14], query vectors for polynomial range-sums have excellent low I/O wavelet approximations, making progressive approximations fast and accurate. As an added benefit, these evaluation methods are complementary to powerful new wavelet-based approximate storage strategies [5].

1.3 Preliminaries

We study evaluation of aggregate queries on a database instance D of a schema F with d numeric attributes ranging from zero to $N-1$. We work with multi-dimensional arrays of real numbers, or functions, indexed by the domain of F, $\mathrm{Dom}(F)$. The set of such functions is a vector space with a natural inner product, $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{\mathbf{x} \in \mathrm{Dom}(F)} \mathbf{a}[\mathbf{x}]\mathbf{b}[\mathbf{x}]$, for any two arrays $\mathbf{a}$ and $\mathbf{b}$. For any set $R \subset \mathrm{Dom}(F)$, χ_R denotes the characteristic function of R: $\chi_R[\mathbf{x}] = 1$ if $\mathbf{x} \in R$ and is zero otherwise.

Following [6, 15], this paper uses the *data frequency distribution* to represent a database as a vector. This is a vector Δ indexed by $\mathrm{Dom}(F)$; for any tuple $\mathbf{x} = (x_0, \ldots, x_{d-1}) \in \mathrm{Dom}(F)$, we define $\Delta[\mathbf{x}]$ as the number of times $\mathbf{x}$ occurs in D. We denote the wavelet transform of a multidimensional array $\mathbf{a}[\mathbf{x}]$ by $\hat{\mathbf{a}}[\xi]$. Because the wavelet transform is invertible, the wavelet transform of the data frequency distribution, $\hat{\Delta}[\xi]$, can be used to evaluate any query on the database D.

This paper studies query evaluation using a version of the wavelet representation stored on disk. One can think of $\hat{\Delta}$ as a materialized view of D. For simplicity, we adopt a simple I/O cost model, even though there are many ways to store and access this view. We assume that the values of $\hat{\Delta}$ are held in either array-based or hash-based storage that allows constant-time access to any single value. We ignore the possibility that several useful values may be allocated on the same disk block, and hence retrieved for free. Similarly, we ignore the potential savings arising from intelligent buffer management. A thorough analysis of wavelet data disk layout strategies and their limits is part of our continuing research. It can be addressed naturally by extending the framework used in this paper.

2. MINIMIZING SSE

In this section we show how wavelets can be used to evaluate a batch of range aggregate queries progressively, minimizing the sum of square errors at each step. This technique is efficient as an exact method, shares I/O optimally, and uses data structures that can be updated quickly. It is based on ProPolyne [14], a technique for the progressive evaluation of single *polynomial range-sum* queries, and reuses much of its machinery. We begin by exhibiting the technique for range Count queries using Haar wavelets. This simple example captures the heart of the matter. Generalization to polynomial range-sum queries using other wavelets is discussed in Section 3.

2.1 Progressive COUNT Queries

Given a rectangular range $R \subset \mathrm{Dom}(F)$, the range Count query $\mathrm{Count}(R, D) = |\sigma_R D|$ simply counts the number of tuples in D that fall in R. Recalling that Δ denotes the data frequency distribution of D, we can write

$$\mathrm{Count}(R, D) = \sum_{\mathbf{x} \in R} \Delta[\mathbf{x}] = \sum_{x \in \mathrm{Dom}(F)} \chi_R[\mathbf{x}]\Delta[\mathbf{x}]$$

In other words, the COUNT query is just the inner product of a (simple) query vector with an (arbitrarily complex) data vector. The Haar wavelets are orthogonal, so the Haar transform preserves inner products. Thus we can write

$$\text{COUNT}(R, D) = \sum_{\xi \in \widehat{\text{Dom}(F)}} \widehat{\chi_R}[\xi]\hat{\Delta}[\xi] \quad (1)$$

giving a formula for query evaluation in the wavelet domain. The functions χ_R are very simple, and it has been shown that $\widehat{\chi_R}$ has at most $O(2^d \log^d N)$ nonzero coefficients, and they can be computed quickly [13, 14, 17]. Thus Equation 1 can be used to evaluate COUNT queries in time $O(2^d \log^d N)$. When storing an entire relation in the wavelet basis, a new tuple can be inserted in time $O(\log^d N)$ [13], making this method competitive with the best known pre-aggregation techniques (see [12] for an overview).

Wavelets have been used successfully for data approximation, and are often thought of as a lossy compression tool. While this approach works to provide an approximate query answering scheme on some datasets, there is no reason to expect than an arbitrary relation would have a good wavelet approximation. We focus instead on using wavelets for *query* approximation. In particular, when evaluating the sum in Equation 1 we add the terms where $\widehat{\chi_R}$ is largest first, expecting the smaller terms to contribute less to the final outcome. At each step, the partial sum we have computed is just the inner product of the data vector with an *approximate query vector*. Experimental results have shown that this particular evaluation order provides accurate, data-independent progressive estimates in practice [14]. As a consequence of Theorems 1 and 2 this evaluation order stands on firm theoretical ground: at each step it minimizes the average error over random data vectors, and it minimizes the maximum possible error.

2.2 Multiple Queries

Now consider the problem of simultaneously evaluating a batch of COUNT queries for ranges $R_0, \ldots, R_{s-1}$ progressively, minimizing SSE at each step. Assume that we have computed and stored the transformed data vector $\hat{\Delta}$ in a data structure that allows constant time access to any value. One simple solution is to use s instances of the single query evaluation technique, and advance them in a round-robin fashion. This turns out to waste a tremendous amount of I/O, since many data wavelet coefficients will be needed for more than one query. This also ignores the fact that some wavelet coefficients may not be tremendously important for any particular query, but are important for the batch as a whole.

We address both of these problems by introducing an I/O sharing technique and showing how it can be adapted to evaluate queries progressively. For each range R_i, compute the list of nonzero wavelet coefficients of χ_{R_i}, merge these lists into a master list, then iterate over this master list, retrieving each needed data coefficient from storage, and using it to advance the computation of each query that needs it. The I/O savings from this technique can be considerable in practice. For example, Section 6 describes one batch of 512 range queries covering 15.7 million records in a database of temperature observations. Using multiple instances of the single query evaluation technique required 923,076 retrievals. Using this I/O sharing technique, all 512 queries were evaluated exactly after 57,456 retrievals.

Note that this requires all of the nonzero wavelet coefficients of the *query function* to be in main memory, but there are relatively few of these when compared to the number of nonzero wavelet coefficients of the data vector. Range-sum queries have very sparse wavelet representations, and this is independent of the underlying database. Nevertheless, it is of practical interest to avoid simultaneous materialization of all of the query coefficients and reduce workspace requirements.

To evaluate queries progressively, we just need to build a heap from the master list that orders wavelets by their "importance" for the query. Instead of iterating over the master list, we repeatedly extract the most important element from this heap, retrieve the corresponding data wavelet coefficient from storage, and use it to advance each of the individual queries in the batch.

But how can we quantify the importance of a wavelet? Since our goal is to control SSE, we consider the importance of a wavelet ξ' to be the worst case error that arises from not retrieving $\hat{\Delta}[\xi']$ on a database with $\sum |\hat{\Delta}[\xi]| = 1$. This worst case will occur when the entire data vector is concentrated at ξ', and $|\hat{\Delta}[\xi']| = 1$. In this situation, ignoring ξ' leads to an SSE of

$$\text{SSE} = \sum_{i=0}^{s-1} |\widehat{\chi_{R_i}}[\xi']|^2 \stackrel{\text{def}}{=} \iota(\xi')$$

which we take as the definition of the *importance function* of ξ'. This definition is intuitive- wavelets that are irrelevant for the batch of queries have zero importance, wavelets that have significant coefficients for many queries will have high importance.

We use this importance function to define BATCH-BIGGEST-B, a progressive query evaluation strategy given in Figure 1. The progressive estimates serve as approximate query results during the computation. Once the computation is complete, they hold the exact query results.

Because it evaluates queries by retrieving the B coefficients with highest importance before retrieving the $(B+1)$th coefficient, this is called a *biggest-B* strategy. Theorems 1 and 2 show that there is no other linear approximation of the batch of queries using B wavelets that has smaller expected or worst case SSE than our B-step progressive estimate. Once size is defined correctly, biggest is in fact best.

3. MORE GENERAL QUERIES

The strategy outlined above works for much more general queries. The critical feature of COUNT queries is that they are *vector queries*: the result is the scalar product of a *query vector* with a *data vector*. Consider the following range aggregate queries for a fixed range $R \subset \text{Dom}(F)$ and how they can be recast as vector queries:

BATCH-BIGGEST-B

Preprocessing: Compute the wavelet transform of the data density function and store with reasonable random-access cost.

Input: a batch of range-sum queries and an importance function.

1. For each query in the batch, initialize its progressive estimate to be zero.
2. Compute the wavelet transform of each query in the batch, and construct a list of its nonzero wavelet coefficients.
3. Merge these lists into a master list.
4. Compute the importance of each wavelet in the master list, and build a max-heap from the set.
5. Extract the maximum element from the heap, retrieve the corresponding data wavelet coefficients, and use it to increment the progressive estimate of each query in the batch according to Equation 2. Repeat until heap is empty.

Figure 1: The Batch-Biggest-B algorithm for progressive evaluation of batch vector queries.

1. COUNT queries

$$\begin{aligned} \text{COUNT}(R, D) &= \sum_{\mathbf{x} \in R} \Delta[\mathbf{x}] \\ &= \sum_{\text{Dom}(F)} \chi_R[\mathbf{x}]\Delta[\mathbf{x}] \\ &= \langle \chi_R, \Delta \rangle \end{aligned}$$

2. SUM queries

$$\begin{aligned} \text{SUM}(R, \mathit{Attribute}_i, D) &= \sum_{\mathbf{x} \in R} x_i \Delta[\mathbf{x}] \\ &= \sum_{\text{Dom}(F)} x_i \chi_R[\mathbf{x}]\Delta[\mathbf{x}] \\ &= \langle x_i \chi_R, \Delta \rangle \end{aligned}$$

3. Sums of products

$$\begin{aligned} &\text{SUMPRODUCT}(R, \mathit{Attribute}_i, \mathit{Attribute}_j, D) \\ &\quad = \sum_{\text{Dom}(F)} x_i x_j \chi_R[\mathbf{x}]\Delta[\mathbf{x}] \\ &\quad = \langle x_i x_j \chi_R, \Delta \rangle \end{aligned}$$

Many other aggregate query results can be derived from vector queries. The three vector queries above can be used to compute AVERAGE and VARIANCE of any attribute, as well as the COVARIANCE between any two attributes. It is pointed out in [16] that a vast array of statistical techniques can be applied at the range level using these three vector queries. For example, it is possible to perform principal component analysis, ANOVA, hypothesis testing, linear regression, and much more.

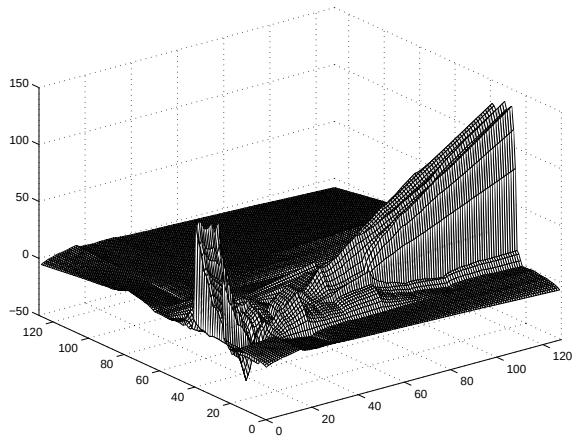

Figure 2: Approximation of a typical Range-Sum query function with 25 Db4 Wavelets

Queries 1 - 3 are very similar. They belong to a powerful class of vector queries that is sufficiently regular to allow efficient query evaluation strategies.

Definition 1. A **polynomial range-sum** of degree δ is a vector query of the form $q[\mathbf{x}] = p(\mathbf{x})\chi_R(\mathbf{x})$ where p is a polynomial in the attributes of F, and R is a hyper-rectangle in $\text{Dom}(F)$, and p has degree at most δ in any variable.

All vector queries can be evaluated in the wavelet domain just as COUNT queries are evaluated using Equation 1. In particular, for any vector query $\mathbf{q}$ and basis of orthogonal wavelets we can write

$$\langle \mathbf{q}, \Delta \rangle = \sum \hat{\mathbf{q}}[\xi]\hat{\Delta}[\xi] = \langle \hat{\mathbf{q}}, \hat{\Delta} \rangle \qquad (2)$$

We now show that this equation can be used to evaluate polynomial range-sums progressively and quickly.

3.1 Progressive Polynomial Range-Sums

We can evaluate polynomial range-sum queries progressively almost exactly as we did in Section 2. The key difference is that we can no longer use Haar wavelets and achieve results efficiently. However, Equation 2 holds for any orthogonal wavelets. It has been shown that using Daubechies wavelets with filter length $2\delta+2$, any polynomial range-sum of degree less than or equal to δ will have less than $((4\delta+2)^d \log^d N)$ nonzero wavelet coefficients. These coefficients can be computed in time $O((4\delta+2)^d \log^d N)$ [14]. Once the data vector has been transformed and stored, new tuples can be inserted in time $O((2\delta+1)^d \log^d N)$. Thus Equation 2 can be used as the basis for an exact polynomial range-sum evaluation strategy that has poly-logarithmic query and update cost. Compare these costs with those for COUNT queries where $\delta = 0$.

As with COUNT queries, polynomial range-sums are approximated very well by a small number of wavelets. To illustrate

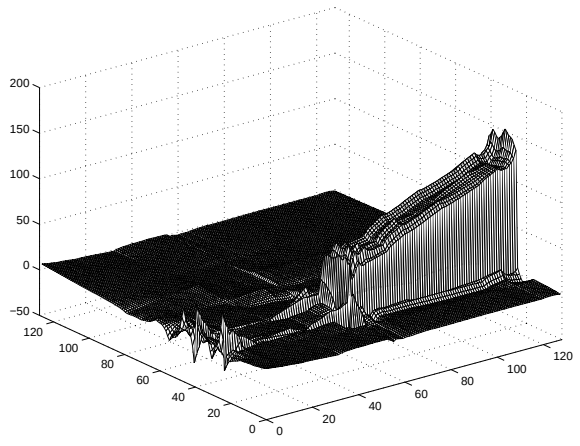

Figure 3: Approximation of a typical Range-Sum query function with 150 Db4 Wavelets

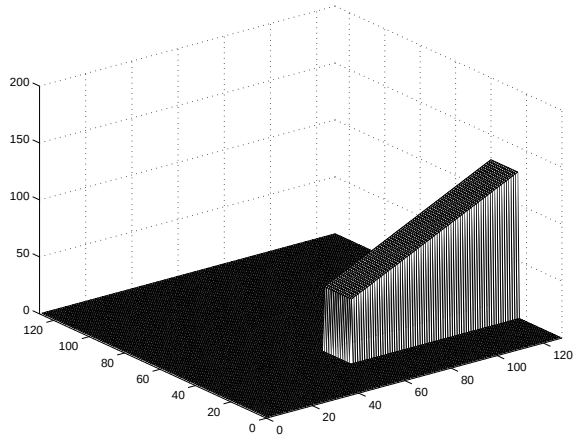

Figure 4: Query function for typical Range-Sum (Computed Exactly using 837 Db4 Wavelets)

this, Figures 2 to 4 display the progressive approximation of a typical degree one polynomial range-sum query vector, $\mathbf{q}[x_1, x_2] = x_1 \chi_R[x_1, x_2]$ where $R = \{(25 \leq x_2 \leq 40) \wedge (55 \leq x_1 \leq 128)\}$. This query function could arise when requesting the total salary paid to employees between age 25 and 40, who make at least 55K per year. Figure 2 displays an approximation using 25 Db4 wavelets. Note that it captures the basic size and shape of the function, but the range boundaries are inexact, and there is some spillover due to the fact that periodic wavelets were used. Figure 3 displays an approximation using 150 wavelets. Here the range boundaries are much sharper, and the periodic spillover smaller. The most striking feature is a Gibbs phenomenon near the range boundaries. Finally, Figure 4 displays the query function reconstructed exactly using 837 wavelets.

3.2 Multiple Polynomial Range-Sums

Now consider the problem of simultaneously evaluating a batch of polynomial range-sums $p_i[\mathbf{x}]\chi_{R_i}[\mathbf{x}], i \in [0, s-1]$. Notice that the definition of the importance function ι in Section 2 did not depend on the fact that we used Haar wavelets. It only depends on the penalty function, SSE. By the same arguments, we can define the importance of a wavelet ξ by

$$\iota(\xi) = \sum_{i=0}^{s-1} |\widehat{p_i \chi_{R_i}}(\xi)|^2$$

and use the algorithm Batch-Biggest-B to evaluate the queries progressively. Theorems 1 and 2 show that this yields the best progressive evaluation strategy for controlling SSE. The experimental results in Section 6 show that this technique produces very accurate answers after retrieving a small number of wavelet coefficients from the database.

4. STRUCTURAL ERROR

Users submit a batch of queries because they are looking for interesting relationships between different query results. Some users may be looking for temporal trends or dramatic jumps. Others may be looking for local maxima. It is often the case that not all of the query results are used at once (e.g., they do not all fit on the screen), and the user is only interested in results that are "near the cursor". In all of these cases, the user is interested in the structure of the query results. If the query results are approximate, the user is concerned with the structure of the error. Errors that impede the task at hand should be penalized, those that are irrelevant should be tolerated if this improves efficiency. This section provides several specific examples of structural error penalty functions, along with formal definitions.

Consider the following scenario. We work with a database of global temperature observations, and the user has partitioned the latitude, longitude and time dimensions into ranges to submit a query in order to find:

Q1 Ranges with the highest average temperatures, *or*

Q2 An accurate view of a set of *high-priority* items currently in use, and a reasonable sketch of the other results, *or*

Q3 Any ranges that are local minima, with average temperature below that of any neighboring range.

An exact result set can be used to provide any of this information, but an approximate result set that works well for one problem may work poorly for another. For example, if each query cell is approximated with no more than 20% relative error, it is possible to do a reasonable job answering Q1, but cannot be trusted to provide meaningful answers for Q3. We now capture this intuition by introducing *structural error penalty functions.*

Definition 2. A **structural error penalty function** is a non-negative homogeneous convex function p on error vectors with the property that $p(0) = 0$ and $p(-x) = p(x)$. As a special case, a **quadratic structural error penalty function** is a positive semi-definite Hermitian quadratic form on error vectors, $p(\epsilon) = \sum P_{ij}\epsilon_i\epsilon_j$.

This definition includes many well known metrics, including L^p norms and Sobolev norms. It is important to allow the

quadratic form to be semi-definite, as it provides the flexibility to say that some errors are irrelevant. We now revisit queries (Q1-Q3) above to see how each one implies a different quadratic penalty function. To simplify the discussion, we assume that the data are dense, so AVERAGE queries reduce to weighted SUM queries.

P1 Minimize the sum of square errors, $p(\epsilon) = \sum |\epsilon[i]|^2$, to allow the user to accurately identify any ranges with a high average temperature.

P2 Minimize a cursored sum of square errors that makes the high-priority cells (say) 10 times more important than the other cells to produce very accurate answers for the high-priority cells, while still substantially penalizing large errors for the remaining cells. $p(\epsilon) = 10 \sum_{i \in H} |\epsilon[i]|^2 + \sum_{i \notin H} |\epsilon[i]|^2$ where H is the set of high-priority ranges.

P3 Minimize the sum of square errors in the discrete Laplacian to penalize any false local extrema.

Linear combinations of quadratic penalty functions are still quadratic penalty functions, allowing them to be mixed arbitrarily to suit the needs of a particular problem.

5. MINIMIZING STRUCTURAL ERROR

An approximate batch query answering technique should aim to minimize an appropriate structural error penalty function. It is impossible to do this on a query-by query basis with a pre-computed synopsis of the data. A pre-computed synopsis must be "tuned" to a particular penalty function and a particular query workload. In this section we demonstrate that using an *online approximation* of the query batch leads to a much more flexible scheme. We introduce a query approximation strategy which finds the "best" query approximation for any penalty function and restriction on the number of records to be retrieved from secondary storage.

One crucial point about the biggest-B progressive query evaluation strategies in Sections 2 and 3 is that the importance function only depends on the penalty function. In fact, SSE is a quadratic penalty function, $p_{\text{SSE}}(\epsilon) = \sum \epsilon_i^2$. Using this notation, we see that the importance function used to control SSE was just $\iota(\xi) = p_{\text{SSE}}(\hat{\mathbf{q}_0}[\xi], \ldots, \hat{\mathbf{q}}_{s-1}[\xi])$ where $\mathbf{q}_i$ are the query vectors for a batch of s queries. In other words, $\iota(\xi)$ is the penalty of the vector $(\hat{\mathbf{q}_0}[\xi], \ldots, \hat{\mathbf{q}}_{s-1}[\xi])$. We can use this definition for any penalty function:

Definition 3. For a penalty function p the **p-weighted biggest-B** approximation of a batch of queries $\mathbf{q}_0, \ldots, \mathbf{q}_{s-1}$ is the progressive estimate given by Batch-Biggest-B after B steps using the importance function

$$\iota_p(\xi) \stackrel{\text{def}}{=} p(\hat{\mathbf{q}}_0[\xi], \ldots, \hat{\mathbf{q}}_{s-1}[\xi])$$

For example, Figure 3 shows the query vector for the SSE-weighted biggest-150 approximation of the query vector depicted in Figure 4. Having defined the strategy we propose to use, we now define the competition.

Definition 4. A **B-term approximation** of a batch of vector queries $\mathbf{q}_0, \ldots, \mathbf{q}_{s-1}$ is an approximation of the batch of queries using only B data wavelet coefficients. That is, given some set Ξ of wavelets with $|\Xi| = B$, the B-term approximation is given by $\langle \mathbf{q}_i, \Delta \rangle \approx \sum_{\xi \in \Xi} \hat{\mathbf{q}}_i[\xi] \hat{\Delta}[\xi]$ for $0 \leq i < s$.

B-term approximations arise from choosing an arbitrary progression order to evaluate a vector query. The theorems that follow state that, in a certain sense, the biggest-B approximations give the best B-term approximations.

5.1 Minimizing the Worst Case

When choosing a progression order to evaluate a batch of queries, it is important to understand how large the structural error penalty can be at each step. The next result shows that biggest-B approximations provide the smallest possible worst-case error, making the progression order used by Batch-Biggest-B optimal. The proof of this result also provides a sharp, easily computed bound on the structural error penalty of any B-term approximation.

Theorem 1. *Assume that $\sum |\hat{\Delta}[\xi]|$ is known. Given a batch of vector queries $\mathbf{q}_0, \ldots, \mathbf{q}_{s-1}$ and a quadratic penalty function p, the worst case penalty for the p-weighted biggest-B approximation is less than or equal to the worst case penalty for any other B-term approximation.*

Proof. For any B-term approximation that uses wavelets in the set Ξ, let ξ' be the most important unused wavelet: $\xi' \notin \Xi$ and $\iota_p(\xi) > \iota_p(\xi')$ implies $\xi \in \Xi$. Now the error vector ϵ is given by $\epsilon = Q_\Xi \hat{\Delta}$ where the matrix Q_Ξ has $Q_\Xi[i, \xi] = \hat{\mathbf{q}}_i[\xi]$ if $\xi \notin \Xi$, and is zero otherwise. Denoting $K = \sum |\hat{\Delta}[\xi]|$, Jensen's inequality allows us to write

$$\begin{aligned}
p(\epsilon) &= p(Q_\Xi \hat{\Delta}) \\
&= p\Big(\sum_{\xi \notin \Xi} |\hat{\Delta}[\xi]| \operatorname{sgn}(\hat{\Delta}[\xi]) Q_\Xi[\cdot, \xi]\Big) \\
&\leq K^{-1} \sum_{\xi \notin \Xi} |\hat{\Delta}[\xi]| p(K Q_\Xi[\cdot, \xi]) \\
&= K^{\alpha - 1} \sum_{\xi \notin \Xi} |\hat{\Delta}[\xi]| p(Q_\Xi[\cdot, \xi]) \\
&= K^{\alpha - 1} \sum_{\xi \notin \Xi} |\hat{\Delta}[\xi]| \iota_p(\xi) \\
&\leq K^\alpha \iota_p(\xi')
\end{aligned}$$

where α is the degree of homogeneity of the penalty function. In fact, equality is obtained when $\hat{\Delta}$ is entirely concentrated on ξ'.

Thus the worst case error for any B-term approximation is given by

$$\mathcal{E}_{\text{worst-case}}(\Xi) = \max_{\xi \notin \Xi} \iota_p(\xi) = K^\alpha \iota_p(\xi')$$

So if Ξ is not a biggest-B strategy, there is an $\eta \in \Xi$ with $\iota_p(\eta) < \iota_p(\xi')$. Substituting ξ' for η in Ξ we obtain a strategy that is at least as good as Ξ. A finite number of these

substitutions produces a biggest-B strategy which is at least as good as Ξ. $\square$

Since norms are examples of structural error penalty functions, Theorem 1 immediately implies the following

COROLLARY 1. *The p-weighted biggest-B approximation minimizes the worst case L^p error for $1 \leq p \leq \infty$.*

In other words, many of the standard penalty functions used to measure the magnitude of vectors can be used with BATCH-BIGGEST-B to minimize the worst case error.

5.2 Minimizing the Average Penalty

Now we turn our attention to analysis of average penalty. When discussing the average penalty of a B-term approximation, we mean the expected penalty of the approximation on a randomly selected database. Surprisingly, it turns out that the best strategy for controlling worst case error is also the best strategy for controlling the average error.

THEOREM 2. *Let data vectors be randomly selected from the set $S^{N^d-1} = \{\Delta \mid \sum \Delta[\mathbf{x}]^2 = 1\}$ with uniform distribution and let $p(\epsilon) = \epsilon^T A \epsilon$ be a quadratic penalty function. The expected penalty incurred by using the p-weighted biggest-B approximation of a query batch is less than or equal to the expected penalty incurred using any other B-term approximation.*

PROOF. Take any B-term approximation that uses wavelets in the set Ξ. For any data vector $\Delta \in S^{N^d-1}$ we write the error vector as in the proof of Theorem 1: $\epsilon = Q_\Xi \hat{\Delta}$.The penalty of this error is given by a quadratic form in $\hat{\Delta}$

$$p(\epsilon_0, \ldots, \epsilon_{s-1}) = \hat{\Delta}^T Q_\Xi^T A Q_\Xi \hat{\Delta} \stackrel{\text{def}}{=} \hat{\Delta}^T R \hat{\Delta}$$

and we can compute the expected penalty by integrating over the unit sphere S^{N^d-1} equipped with the rotation-invariant Lebesgue measure, dP, normalized to make it a probability space. The argument that follows relies on the fact, stated formally in Equation 3, that rotations do not change this measure.

$$E_\Delta[p] = \int_{S^{N^d-1}} \hat{\Delta}^T R \hat{\Delta} \, dP(\Delta)$$

We now use the fact that for any unitary matrix U and any function f on the sphere

$$\int_S f(U\Delta) \, dP(\Delta) = \int_S f(\Delta) \, dP(\Delta) \tag{3}$$

This is just a formal statement of the fact that rotations do not change the measure- the probability is uniformly distributed on the sphere. Noticing the $\hat{\Delta} = W\Delta$ where W is the unitary matrix for the wavelet transform, Equation 3 we write

$$E_\Delta[p] = \int_{S^{N^d-1}} \hat{\Delta}^T R \hat{\Delta} \, dP(\hat{\Delta})$$

Because R is symmetric, it can be diagonalized. There is a diagonal matrix D and a unitary matrix U such that $R = U^T D U$. Using Equation 3 once more, we see that

$$\begin{aligned} E_\Delta[p] &= \int_{S^{N^d-1}} \hat{\Delta}^T D \hat{\Delta} \, dP(\hat{\Delta}) \\ &= \sum_{i=0}^{N^d-1} D[i,i] \int_{S^{N^d-1}} |\hat{\Delta}[i]|^2 \, dP(\hat{\Delta}) \\ &= (N^d - 1)^{-1} \operatorname{trace}(D) \end{aligned}$$

where the last equality follows from the fact that on any n-sphere

$$\int_{S^n} x_i^2 \, dP = \frac{1}{n} \sum_{k=0}^{n} \int_{S^n} x_k^2 \, dP = \frac{1}{n}$$

Now D and R are similar matrices, $\operatorname{trace}(D) = \operatorname{trace}(R)$, so

$$E_\Delta[p] = (N^d - 1)^{-1} \operatorname{trace}(R)$$

By the definition of R, $\operatorname{trace}(R) = \sum_{\xi \notin \Xi} \sum_{i,j} \hat{\mathbf{q}}_i[\xi] A_{ij} \hat{\mathbf{q}}_j[\xi]$, which is clearly minimized by placing those ξ where $\sum_{i,j} \hat{\mathbf{q}}_i[\xi] A_{ij} \hat{\mathbf{q}}_j[\xi]$ is largest in Ξ. This is exactly what the p-weighted biggest-B strategy does. $\square$

We use the two theorems above to justify the statement that *the biggest-B approximation is also the best-B approximation.* Notice also that the proof of Theorem 1 provides a guaranteed upper bound for observed error penalty. The proof of Theorem 2 provides an estimate of the average penalty.

6. EXPERIMENTAL RESULTS

We have implemented BATCH-BIGGEST-B for polynomial range-sums and tested it on empirical datasets. We present a sample of results from trials on a dataset of temperature observations around the globe at various altitudes during March and April 2001. The dataset has 15.7 million records and 5 dimensions: latitude, longitude, altitude, time, and temperature. The queries executed partitioned entire data domain into 512 randomly sized ranges, and sum the temperature in each range. The key observations follow.

Observation 1: I/O sharing is considerable. The query batch partitioned the entire data domain. In order to answer the queries directly from a table, all 15.7 million records would need to be scanned. The Db4 wavelet representation of this dataset has over 13 million nonzero coefficients. Repeatedly using the single-query version of ProPolyne requires the retrieval of 923076 wavelet coefficients, approximately 1800 wavelets per range. Using BATCH-BIGGEST-B to compute these queries simultaneously (and exactly) requires the retrieval of 57456 wavelets, approximately 112 wavelets per range. Thus BATCH-BIGGEST-B provides an efficient exact algorithm by exploiting I/O sharing across queries.

In each of these cases notice that only a small fraction of the data wavelet coefficients need to be retrieved. This has nothing to do with the underlying dataset. It is simply a quantification of the fact that ProPolyne is an efficient exact

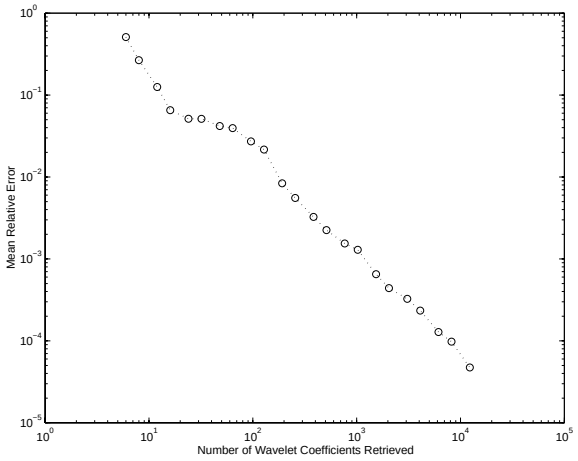

Figure 5: Progressive mean relative error for progression minimizing SSE.

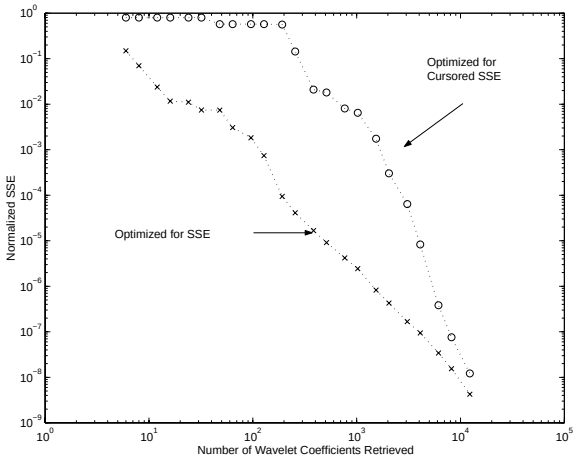

Figure 6: Progressive SSE Penalty for two progressive query strategies.

algorithm- most of the data wavelet coefficients are irrelevant for the query. Another way to say this is that most of the *query* wavelet coefficients are zero. See [14] for details.

As mentioned in Section 1.2, Batch-Biggest-B can be used with other transformation based techniques. Using prefix-sums to compute each of these queries requires the retrieval of 8192 precomputed values. When using Batch-Biggest-B to share this retrieval cost between ranges, only 512 prefix-sums are retrieved.

Observation 2: Progressive estimates become accurate quickly. Figure 5 shows the mean relative error of progressive estimates versus the number of coefficients retrieved. We see that after retrieving only 128 wavelets, the mean relative error is below 1%. Note the log scale on both axes. *Thus we have accurate results after retrieving less than one wavelet for each submitted query.*

Observation 3: Choosing the right penalty function makes a difference. Our implementation accepts arbitrary quadratic

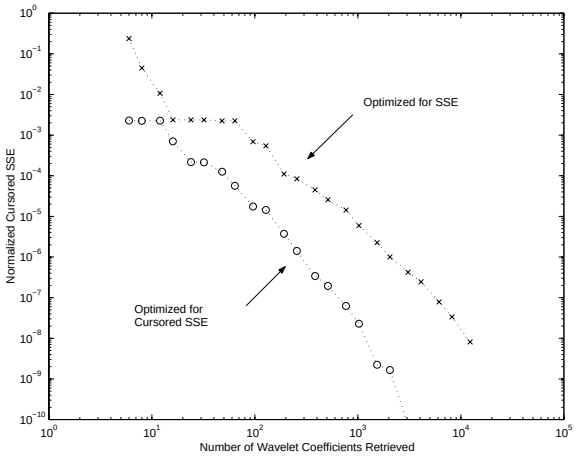

Figure 7: Progressive cursored SSE Penalty for two progressive query strategies.

penalty functions. Figures 6 and 7 display progressive results of query evaluation when minimizing SSE and when minimizing a cursored SSE that prioritizes errors in a set of 20 neighboring ranges 10 times more than it penalizes errors in the other ranges. The horizontal axis of each chart displays the number of wavelets retrieved. Figure 6 shows the normalized SSE for each of these runs. Normalized SSE is the SSE divided by the sum of squares of query results. We see that the trial minimizing SSE has substantially and consistently lower SSE than the trial minimizing the cursored SSE. Figure 7 displays the normalized cursored SSE for the same two trials. Here we see that the trial minimizing the cursored SSE leads to substantially lower penalty.

7. CONCLUSION

This paper presented a framework for progressive answering of multiple range-sum queries. Two major problems arise in this setting that are not present when evaluating single range-sums: I/O should be shared as much as possible, and the progressive approximation should control the structure of the error according to a penalty function specified by the user. We addressed both of these issues with the introduction of Batch-Biggest-B and the proof in Theorems 1 and 2 that using the penalty function to weigh the importance of retrieving a particular item from storage leads to optimal estimates.

These results immediately raise several questions. Foremost among these is the need to generalize importance functions to disk blocks rather than individual tuples. Such a generalization is a step in the development of optimal disk layout strategies for wavelet data. Combining this analysis with workload information will lead to techniques for smart buffer management. We also plan to investigate the use of these techniques to deliver progressive implementations of relational algebra as well as commercial OLAP query languages. Finally, we want to study the limits of linear algebraic query evaluation and approximation techniques. In particular, it would be interesting to know whether or not it is possible to design transformations specifically for the range-sum problem that perform significantly better than the wavelets used here.

8. ACKNOWLEDGMENTS

The authors would like to thank Anna Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss for their insightful discussions and comments on drafts of this paper. We would like to thank our colleagues George Hajj and Brian Wilson at JPL for providing the temperature dataset.

9. REFERENCES

[1] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 111–122, 2000.

[2] C.-Y. Chan and Y. E. Ionnidis. Hierarchical cubes for range-sum queries. In *VLDB 1999, Proc. of 25th International Conference on Very Large Data Bases*, pages 675–686, 1999.

[3] S. Geffner, D. Agrawal, and A. E. Abbadi. The dynamic data cube. In *EDBT 2000, 7th International Conference on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 237–253. Springer, 2000.

[4] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Optimal and approximate computation of summary statistics for range aggregates. In *PODS 2001, Proceedings of the Twenteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 228–237, 2001.

[5] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases*, 2001.

[6] D. Gunopulos, G. Kollios, V. J. Tsotras, and C. Domeniconi. Approximating multi-dimensional aggregate range queries over real attributes. In *SIGMOD 2000, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 463–474, 2000.

[7] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 171–182. ACM Press, 1997.

[8] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 73–88. ACM Press, 1997.

[9] I. Lazaridis and S. Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD 2001, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 401–412, 2001.

[10] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *SSDBM 1999, 11th International Conference on Scientific and Statistical Database Management*, pages 24–33. IEEE Computer Society, 1999.

[11] M. Riedewald, D. Agrawal, and A. E. Abbadi. pCube: Update-efficient online aggregation with progressive feedback. In *SSDBM 2000, Proceedings of the 12th International Conference on Scientific and Statistical Database Management*, pages 95–108, 2000.

[12] M. Riedewald, D. Agrawal, and A. E. Abbadi. Flexible data cubes for online aggregation. In *ICDT 2001, 8th International Conference on Database Theory*, pages 159–173, 2001.

[13] 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.

[14] R. R. Schmidt and C. Shahabi. Propolyne: A fast wavelet-based technique for progressive evaluation of polynomial range-sum queries. In *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology*, Lecture Notes in Computer Science. Springer, 2002.

[15] J. Shanmugasundaram, U. Fayyad, and P. Bradley. Compressed data cubes for OLAP aggregate query approximation on continuous dimensions. In *Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 1999.

[16] S.-C. Shao. Multivariate and multidimensional OLAP. In *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1998.

[17] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD 1999, Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 193–204. ACM Press, 1999.

[18] Y.-L. Wu, D. Agrawal, and A. E. Abbadi. Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes. In *CIKM 2000, Proceedings of the 9th Interntationall Conference on Information and Knowledge Management*, pages 414–421. ACM, 2000.

[19] Y. Zhao, P. Deshpande, J. F. Naughton, and A. Shukla. Simultaneous optimization and evaluation of multiple dimensional queries. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 271–282. ACM Press, 1998.