Wavelet Disk Placement for Efficient Querying of Large Multidimensional Data Sets

Cyrus Shahabi
University of Southern California
Department of Computer Science
Los Angeles CA 90089-0781, USA
email: shahabi@usc.edu

Rolfe S. Schmidt
Cedars-Sinai Medical Center
Louis Warschaw Prostate Cancer Center
Los Angeles CA 90048, USA
email: rolfe.schmidt@cshs.org

March 21, 2003

Abstract

New data-intensive applications operate on diverse types of data, with new characteristics in querying the data. In particular: 1) the data set is large and multidimensional; popular examples are spatial and temporal data, as well as sensor data streams, 2) the queries are complex, asking for trends or outliers in data, correlation between different dimensions, or aggregation of one or more (measure) attributes given a bounded multidimensional space (termed, range-aggregate queries), and 3) the applications are online and interactive requiring fast response time and hence the results can be approximate and/or progressively become exact. These characteristics lead us to believe that wavelet transform will become a likely tool for future database query processing. Although a straightforward adoption of wavelets is to utilize it to reduce the data size at different resolutions, unfortunately data compression methods are only effective on datasets that compress well and on queries that require the reconstruction of the entire signal.

Therefore, we are taking a fundamentally different approach. Our approach is a data independent approximation technique that is based on query approximation rather than data compression. Many common queries, including relational algebra expressions and a large class of aggregate queries, can be conceptualized as performing a linear transformation on the density distribution of an input relation to produce the density of an output relation. This observation leads us to the following generic progressive query evaluation plan: decompose the query transformation into a series of small, cheaply computable sub-transformations and evaluate the most important sub-transformations first. In this paper, we discuss the details of this evaluation plan for polynomial range-sum queries, and show how it can be extended to support a batch of several submitted queries. In addition, we show that typical queries on wavelet data require a distinct access pattern which we describe and exploit to design a disk placement strategy for wavelet data that yields best-possible I/O complexity for point and range query evaluation. We conclude by discussing some open problems in dealing with wavelets from a database perspective such as how to perform I/O efficient wavelet transformation.

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and IIS-0082826, NASA/JPL contract nr. 901518, and unrestricted cash gifts from Okawa Foundation, Microsoft, and NCR.
1 Introduction

Since the introduction of the fast wavelet transform [11, 26], multi-resolution analysis has proven to be a powerful tool for a wide range of applications. Wavelets often bring to mind applications like signal and image processing, not databases. But in order to scale these applications to very large datasets, or to provide access to multiple users, it is important to treat the storage and retrieval of wavelet data as a database problem.

Actually, recent work suggests even stronger reasons for us to be interested in management of multi-resolution data. By storing a wavelet representation of a relation or a data cube instead of a tabular representation, one can provide fast approximate [40], exact, and progressive [37, 36] range aggregate query support. So multi-resolution data may play a role in general purpose database systems. This should not be surprising: since their beginning Online Analytical Processing (OLAP) systems have used dimension hierarchies to store what amounts to a multi-resolution view of a data cube.

Traditionally, data transformation techniques such as wavelets have been used to compress data. The idea is to transform the raw data set to an alternative form, in which many data points (termed coefficients) become zero or small enough to be negligible exploiting the inherent redundancy in the raw data set. Consequently, the negligible coefficients can be dropped and the rest would be sufficient to reconstruct the data later with minimum error and hence the compression of data.

However, there is a major difference between the main objective of signal processing and compression applications using wavelets and that of database applications. With compression applications, the main objective is to compress data in such a way that one can reconstruct the data set in its entirety with as minimal error as possible. Consequently, at the data generation time, one can decide which wavelets to keep and which to drop. Instead, with database queries, each range-sum query is interested in some bounded area (i.e., subset) of the data. The reconstruction of the entire signal is only one of the many possible queries.

Hence, for the database applications, at the data generation or population time, one cannot optimally specify which coefficients to keep or drop. We believe that for queries to observe lower variance in the accuracy of their results, the decision of which coefficients are important must be delayed to the query time. This of course means that all the data coefficients must be kept with no compression at the data population time. This is justified since the price of storage is low and decreasing. Even the traditional OLAP compression techniques [40] did not have the concern of saving space but the implicit objective of improving query response time by dealing with smaller size datasets. Towards this end, we propose a technique that use wavelets to approximate incoming queries rather than the underlying data¹.

¹Note that the data set is still transformed using wavelet; however, it is not approximated since we keep all the coefficients.
1.1 Prior Work

For the past two years, we have been investigating efficient techniques to support range-sum queries on large multidimensional data sets. We have introduced a 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 techniques (see [37]). We achieve this by observing that polynomial range-sums can be translated and evaluated in the wavelet domain. When the wavelet filter is chosen to satisfy an appropriate moment condition, most of the query wavelet coefficients vanish making the query evaluation faster. We made this observation practical by introducing the lazy wavelet transform, an algorithm that translates polynomial range-sums to the wavelet domain in poly-logarithmic time.

Note that with our lazy wavelet transform, the cost of the range-sum query evaluation becomes independent of the size of the range. To illustrate the main intuition behind our lazy wavelet transform, assume Haar wavelet on a one-dimensional count query with domain size of \( N \). Here, at any recursive step in the Haar transform of a constant count function the summary coefficients are constant within 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 \( \log N \) steps can be carried out in constant time, allowing us to perform the entire transform in time and space \( \Theta(\log N) \). This is while 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. In [37], we have formalized these arguments and extended them to deal with general \( d \) 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 \( O(\log N) \). We have shown that with appropriate restrictions and choices of wavelet filters, we can obtain both of these features for general polynomial range-sums.

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 brought these ideas together by introducing ProPolyne (Progressive Polynomial Range-Sum Evaluator), 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 (not only COUNT, SUM and AVERAGE, but also VARIANCE, COVARIANCE and more). All computations are performed entirely in the wavelet domain.

2. Uses the lazy wavelet transform to achieve query and update cost comparable to the best known exact techniques.

3. By using the most important query wavelet coefficients first, provides excellent approximate results and guaranteed error bounds with very little I/O and computational
overhead, reaching low relative error far more quickly than analogous data compression methods.

In a later publication [36], we focused on applications that require simultaneous evaluation of a batch of range aggregate queries. 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.

Consequently, we have extended ProPolyne to exploit I/O sharing across queries in a batch to provide fast exact results for the entire batch. Another major contribution of [36] has been 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. As a result, we have developed a progressive version of ProPolyne-batch queries that can accept any penalty function specified at the query time, and minimize the worst case and average penalty at each step of the computation.

ProPolyne has been used on several empirical data sets such as the ground sensor data [37], atmospheric data [36], petroleum sale data [3] and immersive sensor data [38]. Our experimental results on these empirical datasets showed that the approximate results produced by ProPolyne are very accurate long before the exact query evaluation is complete. These experiments also showed 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.

1.2 New Contributions

As mentioned before, ProPolyne is for applications that deal with large amount of wavelet data that cannot fit in main memory. Hence, the challenge is how to optimally store the wavelets on the secondary storage, i.e., magnetic disk drives. Thanks to the principle of locality of reference, we often find that when an application needs to access one datum on a disk block, it is likely to need to access other data on the same block. By designing applications to take advantage of this, we can amortize the cost of disk access over multiple reads, significantly reducing the total I/O cost. Note that in order to realize these savings we must both design the consumer to be disk block aware, and ensure that we lay the data out on the disk in a way that makes the principle of locality of reference hold.

Is there a principle of locality of reference for wavelet data? Or more precisely, is there a way we can store wavelet data to create such a principle? In this paper we show that
we can, and that for common access patterns we have a much stronger principle. It turns out that for ProPolyne point and range queries, if a wavelet coefficient is retrieved, we are guaranteed that all of its dependent coefficients will also be retrieved. The challenge is that distinct coefficients will have common dependents. In order to make applications that rely on access to wavelet data scalable, we must take full advantage of this unique access pattern. This is the goal of our paper.

In this paper we describe the access patterns required for processing point and range queries on wavelet data. We study the space of all possible (non-redundant) allocations of these data to disk blocks. Our major contributions include

- An allocation of wavelet coefficients to disk blocks of size $B$ so that if at least one item on the block is needed to answer a point query, then a total of $\lfloor \lg B \rfloor$ items on the block will be needed (Theorem 1).

- A proof that the allocation of Theorem 1 is essentially optimal: for all disk block of size $B$, if the block must be retrieved to answer a query, the expected number of needed items on the block is less than $1 + \lg B$ (Theorem 2).

- An extension of these results to multi-dimensional data and range queries (Theorem 3).

- Definition of a query dependent importance function on disk blocks which allows us to perform the most valuable I/O's first and deliver excellent approximate results progressively during query evaluation (Theorem 4).

We find these results and analysis theoretically satisfying, but the work solves a very practical problem that arose in our development of the ProPolyne system. We show experimental results in Section 8 that demonstrate quantitatively how important block allocation is for efficient ProPolyne query answering, and that some apparently natural strategies perform three to four times worse.

The remainder of this paper is organized as follows. In Section 2, we discuss the related work. Section 3 briefly provides the required background on ProPolyne multi-resolution queries. We then first discuss our techniques assuming one dimensional data in Sections 4 and 5. Subsequently, we show how the techniques can be generalized to multi-dimensions in Section 6. In Section 7, we explain how ProPolyne can be extended to work with disk blocks rather than individual coefficients. Section 8 reports on our experimental results. Finally, we conclude in Section 9 and discuss some open problems in Section 10.

2 Related Work

Extensive research has been done to find efficient ways to evaluate range aggregates. The prefix-sum method presented in [20] 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 [15, 14, 33]. Hierarchical Cubes [7] build on the ideas of [15] to provide configurable
tradeoff between query and update cost, as do Iterative Data Cubes (IDC) [34]. In fact, IDC generalizes the techniques of [15, 14, 33, 20], and certain forms of the Hierarchical Cubes. IDC is the pre-aggregation technique most closely related to our work: one of the variation of ProPolyne with fixed measure falls into this framework, but our density-based algorithms do not. Even the fixed-measure version of ProPolyne is more than an IDC, it also provides data independent progressive query evaluation.

Approximate query answering has been proposed as a way to obtain even faster results. Histograms [30, 22, 2, 29, 12, 5, 21, 19] and sampling [8, 10, 44] for selectivity estimation have a rich literature, and provide adaptive, optimizable data compression techniques for query answering [9, 9, 17]. There has also been work in modeling for approximate query answering [4, 16]. In [39] clustering and mixed Gaussian estimators are used to approximate the data density function for approximate query support. The density function is also exploited by [19, 35]. The flexible measures supported by this approach inspire its use in this study. Recently researchers have addressed progressive query answering [32, 43, 24]. These techniques, whether tree based or multi-resolution analysis based, share a common strategy: answer queries quickly using low resolution information, and recursively refine the result with higher resolution data. At each resolution level, these techniques must retrieve all summary nodes that overlap the boundary of the range. This makes the final worst-case complexity proportional to the surface area of the range, limiting their utility as exact algorithms. ProPolyne is fundamentally different: it starts by extracting information from the resolution levels that are most relevant for the submitted query. ProPolyne also has the benefit that at each resolution level, it only needs to retrieve summary statistics with domains overlapping the corners of the range, giving it excellent performance as an exact algorithm.

Recently wavelets have emerged as a powerful tool for approximate answering of aggregate [17, 40, 41, 27, 43] and relational algebra [6] queries. Streaming algorithms for approximate population of a wavelet database are also available [18], making wavelet coefficients a powerful approximate data storage format. Most of the wavelet query evaluation work has focused on using wavelets to compress the underlying data, reducing the size of the problem. A notable exception is [17] which proposes a method to approximate the function that maps ranges to the corresponding range-sum, simultaneously approximating all SUM queries for a given measure. This method is the closest in spirit to the techniques we present; besides supporting a different class of queries, our technique differs by approximating individual queries at the time of submission, rather than approximating all queries at the time of database population.

The wavelet disk placement work arose out of our recent efforts to use wavelet-based operator approximation for approximate query answering, aka ProPolyne [37, 36]. While studying these methods, it became clear to us that efficient disk access would be necessary for any practical system. All of the data approximation techniques discussed above assume that the compressed dataset will fit in main memory or be scanned from disk in its entirety. To our knowledge, efficient disk placement of wavelet data has not been explored before this work.

Another area of seemingly related work involves the use of space-filling curves, z-
ordering, and Gray codes to place a one dimensional index on multi-dimensional data that still clusters data spatially [28]. In fact, this is more closely related to our future work on storage of sparse wavelet data, and have no need for index structures. With this said, we do compare the performance of our techniques with a domain slicing block allocation scheme that is very similar to (and sometimes identical to) space-filling curve based allocations. In our experiments we find that our optimal block placement strategy requires three to four times less disk access than the domain slicing method.

3 ProPolyne Background

We begin by making our problem precise. Haar wavelets provide an orthonormal basis for the vector space of all functions on a data domain. See [42] for a thorough introduction to wavelets and related algorithms, and see [31] for a concise introduction to compactly supported wavelets such as the Haar wavelets used in this paper. It is worth noting that our results hold for any wavelet filter that satisfies certain technical conditions discussed in [37]. However, to simplify the presentation of the paper and to focus on the main contributions, we use Haar wavelets throughout the paper.

We denote the data domain by \( D \), and assume that it is a \( d \)-dimensional lattice. When we store wavelet coefficients, we are really storing a representation of a function. This makes a particular type of query very natural:

**Definition 1** A **point query** on a dataset containing a representation of a function \( f : D \to \mathbb{R} \) specifies a point \( x \in D \), and receives \( f(x) \) as an answer.

A relation can be represented by its data density or measure density function. When this is done, a large class of range aggregate queries are seen to be inner products of **query functions** with **data functions** in the function space [37]. Specifically, it is possible to support traditional aggregate functions such as COUNT, SUM, and AVERAGE, as well as less traditional aggregates including COVARIANCE, REGRESSION-SLOPE, and KURTOSIS as long as we can support the following basic range query:

**Definition 2** A **polynomial range-sum** on a dataset containing a representation of a function \( f : D \to \mathbb{R} \) specifies a range \( R \subset D \) and a polynomial \( p \) on the coordinates of points in \( D \), and receives \( \langle f, p_{|R} \rangle = \sum_{x \in R} p(x)f(x) \) as an answer.

Which wavelet coefficients do we need to retrieve from storage in order to answer these queries? For the point query the answer is simple- the only wavelet coefficients needed to reconstruct the value of a function at a point \( x \) are those corresponding to wavelets that are not zero at \( x \). In other words, we are only interested in wavelets whose support overlaps \( x \).

One of the fundamental observations of ProPolyne [37] is that (when certain technical conditions are satisfied) the only wavelets that are relevant for answering a polynomial
range-sum query are those whose support overlaps a corner of the range $R$. Thus in a $d$-dimensional domain, a polynomial range-sum query requires the same disk access as $2^d$ point queries. In this paper we are only interested in efficient disk access, not in the computations that occur afterwards. From this perspective both point and range queries can be distilled to a more fundamental selection query.

**Definition 3** The **wavelet overlap query** on a dataset of wavelet coefficients specifies a point $x \in D$ and returns the set of all wavelet coefficients for wavelets whose support overlaps $x$. Denoting this query by $WQ(x)$, we write

$$WQ(x) = \{a_{j,k} | \psi_{j,k}(x) \neq 0\}$$

where $a_{j,k} = \langle f, \psi_{j,k} \rangle$ denotes the wavelet coefficient of the stored function $f$ at resolution level $j$ and offset $k$.

This is the fundamental query of interest when storing data in any wavelet basis, but for the particular case of one-dimensional Haar wavelets, we have an explicit definition of the set:

$$WQ(x) = \{a_{j,k} | 2^j k \leq x < 2^j (k + 1)\}$$

Our goal is to evaluate these queries with as little disk access as possible.

## 4 A Framework for One Dimension

Before we can find the best possible disk allocation, we need a precise notion of what a disk allocation is and how access patterns for wavelet overlap queries determine which blocks will be retrieved. The purpose of this section is to make these notions precise.

In particular, we observe that our access patterns can be captured by the **wavelet dependency graph**, a directed acyclic graph whose leaf nodes correspond to points in the data domain, and whose internal nodes correspond to wavelets. The key observation is that to answer $WQ(x)$ we need to retrieve exactly the wavelet coefficients corresponding to nodes on the graph reachable from the leaf $x$. With this framework, an allocation of wavelet data to disk blocks corresponds to a tiling of the (internal nodes of the) dependency graph, and the I/O cost of a wavelet overlap query is just the number of tiles reachable from a leaf node in the graph.

Throughout this paper we assume that a disk block holds $B$ wavelet coefficients, and has a unique identifier (e.g., its physical address). With this we define

**Definition 4** A **block allocation** for a collection of wavelet coefficients is a $B$ to one function from wavelets to disk block identifiers. In other words, it is an assignment of wavelets to disk blocks.

To help us reason about different block allocations, we try to capture the essence of our access patterns with the following
Definition 5 The dependency graph for Haar wavelets on a domain $D$ is the directed acyclic graph $G = (V, E)$ defined as follows. Let $\hat{D}$ denote the set of all Haar wavelets on $D$, and let $V = D \cup \hat{D}$. In other words the vertices of the graph are either points in the domain or wavelets on the domain. The edges are defined by the following rules

- For two wavelets $\xi, \eta$, the pair $(\xi, \eta) \in E$ if and only if $\eta(x) = 0$ implies that $\xi(x) = 0$ (the interval where $\xi$ "lives" is contained in the interval where $\eta$ "lives").

- For $x \in D$, $\xi \in \hat{D}$, the pair $(x, \xi) \in E$ if and only if $\xi(x) \neq 0$ and there is no wavelet $\eta$ such that $(\eta, \xi) \in E$.

- For two points $x, y \in D$, $(x, y) \notin E$

For a one-dimensional domain, this graph is a tree.

This definition is cumbersome but explicit. To make this more concrete, an example dependency graph is depicted in Figure 1 for Haar wavelets on a domain of size eight. At each node in the figure we depict the graph of the corresponding wavelet. The topmost node corresponds to the constant function, and wavelet coefficients for this node are just averages over the entire domain. At the next level we have a low frequency wavelet whose support covers the entire domain. As we move to higher resolution levels, the size of the support is halved, and the "frequency" is doubled. In one dimension, the dependency graph is isomorphic to the error tree [40].

There is another characterization of this graph which is very interesting to us.

Claim 1 The dependency graph for Haar wavelets is the minimal graph $G$ with vertices $V = D \cup \hat{D}$ such that if $\xi_1$ is needed to answer $WQ(x)$ and $\xi_2$ is reachable from $\xi_1$, then $\xi_2$ is also needed to answer the query. In particular $WQ(x) = \{a_\xi \mid \text{there exists a path from } x \text{ to } \xi\}$.
In other words, using this graph turns our wavelet overlap query into a reachability query on a dag. We can also replace block allocations with vertex tilings.

**Definition 6** A tiling for a graph $G = (V, E)$ with tiles of size $B$ is a partition of the vertices into disjoint tiles $T_i$ with $|T_i| \leq B$.

There is clearly a one to one correspondence between tilings of (the internal nodes of) the dependency graph and block allocations. Moreover, this gives us a simple definition of the I/O cost of a wavelet overlap query.

**Claim 2** For a given block allocation, the number of disk blocks that must be retrieved to answer a wavelet overlap query $WQ(x)$ is equal to the number of tiles reachable from $x$ in the tiling corresponding to the block allocation.

So to reduce the I/O cost of answering wavelet overlap queries, we should find tilings that are “minimally reachable”. Intuitively, it seems that we will be doing well if we can simply ensure that whenever a tile is reachable from a point, it has many vertices that are reachable from that point. The number of vertices reachable from a point is independent of the tiling. If each reachable tile covers a large number of the reachable vertices, then all reachable vertices will be covered with a small number of tiles. So it seems that a particular tile is good if it has a high usage rate.

**Definition 7** For a dag $G = (V, E)$, a tile $T \subseteq V$, and a source $s \in V$, the usage rate of $T$ at $s$ is

$$u(T, s) = |\{v \in T \mid v \text{ is reachable from } s\}|$$

Denote the number of sources that can reach $T$ by $S(T)$. The average usage of $T$ is

$$u(T) = \frac{1}{S(T)} \sum_{s \in \text{sources}(G)} u(T, s)$$

(1)

So if wavelet overlap queries are generated randomly with the uniform distribution over the data domain, the average usage of a tile corresponds to the expected number of items to be used on a disk block, given the fact that the block had to be retrieved. Example tiles with usage rates can be seen in Figure 2. There is a closely related function that indicates whether a tile is reachable from a particular source.

**Definition 8** For a dag $G = (V, E)$, tile $T \subseteq V$, and a source $s \in V$ the requirement function $r(T, s)$ is 1 if some element of $T$ is reachable from $s$ ($u(T, s) > 0$), and is zero otherwise.

Notice that

$$S(T) = \sum_{s \in \text{sources}(G)} r(T, s)$$

(2)
While designing tiles with maximal usage seems like a good intermediate goal, eventually we want to minimize average I/O cost for query answering. This corresponds to minimizing the average number of tiles reachable from a randomly selected source. Thus we define the \textit{cost} of a source $s$ for a tiling $T$ as the number of tiles reachable from $s$

$$c(s) = \sum_{T \in T} r(T, s)$$

We will actually be able to show in Section 5 that maximizing usage is the same as minimizing cost, a result that relies on the following equation, which is valid for any source $s$ in the one dimensional Haar wavelet dependency graph.

$$\log |D| = \sum_{T \in T} u(T, s)$$

This holds because both sides represent the total number of non-source vertices reachable form $s$.

\section{Optimal Placement in One Dimension}

With this terminology in hand, we can start solving problems. We begin by computing the usage rates of two tile types.

\textbf{Example 1 (Straight-line Tiles)} When answering a wavelet overlap query for a point $x \in D$, it would be ideal if every tile we reached were used entirely. Recalling that for the one-dimensional case, the dependency graph is a binary tree, we see that we can achieve this by choosing “straight-line” tiles of the form $T = \{v, \text{parent}(v), \text{parent}(\text{parent}(v)), \ldots, \text{parent}^{B-1}(v)\}$ so that if there is a path from $s$ to $v$, then $s$ reaches every element in the tile and $u(T, s) = B$. An example of a straight-line tile of size three in a domain of size eight can be seen in Figure
Note that by including $\xi_1$ in this tile, we do very well in answering the point query at points 0 or 1, but do quite poorly when answering a point query at 4, 5, 6, or 7. We can formalize this observation.

Notice that the number of sources that can reach $v$ is exactly one half of the number of sources that can reach $\text{parent}(v)$ because $\text{parent}(v)$ is a wavelet that lives on an interval double the size of $v$. Thus for every source that gets the optimal usage out of $T$, there is another one that obtains one less than the optimal usage.

So far this may not seem discouraging, but the same argument says that exactly one half of all sources that reach $T$ reach it only at the highest node, $\text{parent}^{B-1}(v)$, and obtain the worst possible usage rate of one. This lets us write a recursion for the usage rate of straight-line tiles of (now variable) length $B$

$$U_{\text{straight-line}}(B) = 1 + \frac{U_{\text{straight-line}}(B-1)}{2} < 2$$

Thus no matter what the tile size, if $T$ is a straight line tile, $u(T) < 2$. Unless we find something better it will be very difficult to provide scalable access to wavelet data.

Example 2 (Dense Subtree Tiles) Because the straight-line tiles worked so poorly, let us go to the other extreme and look at connected tiles that are as short as possible. For convenience, assume that $B = 2^k - 1$ for some natural number $k$. Then for any vertex $v$ that is at least $\log B$ steps away from any source in the dependency graph, we can consider the tile $T = \{v' \in V | v = \text{parent}^j(v'), j < \log B\}$, which is a dense binary tree. An example tile of size three can be seen in Figure 2.

It is particularly easy to compute the average usage of this tile. In fact the usage rate of this tile is the same from any vertex, and is equal to $[\log B]$. Thus $u(T) = [\log B]$. If $B$ is not of the special form assumed above, we may not be able to complete the bottom level of the dense binary tree, but we can be assured that our usage rate will always be at least $[\log B]$.

It is easy to cover any given dependency graph with these dense connected subtiles (perhaps running out of room at the bottom and settling for “connected-as-possible” tiles). An alternative approach is to map this problem to the problem of finding an optimal $B$-partitioning of a weighted graph/tree [23, 25] where $B$ is the size of a disk block. In this case, weight of 1 should be assigned to each node of the dependency graph and the weights on its edges should capture $u(T)$. One way to achieve this is by assigning each edge the weight of the number of leaves reachable down of that edge. Regardless of how we generate the covering, it leads directly to a proof of Theorem 1.

Theorem 1 A binary tree with $N$ leaf nodes has a tiling such that for every source $s \in V$ and every tile $T$ which does not contain a source node has $u(T, s) \geq [\log B]$.

A useful corollary is

Corollary 1 Using the block allocation of Theorem 1 on a domain of size $N$, one must retrieve no more than $1 + \log B N$ disk blocks to answer a wavelet overlap or point query. Range aggregate queries require no more than twice this amount.
Recall that as discussed in Section 1.1, the cost of a range query is independent of the size of the range since we only need to compute wavelets at the boundaries of the range. While this is clearly a better situation than what we saw with our straight-line blocks, we still might hope for more. Logarithmic usage rates seriously restrict our ability to amortize I/O cost by using large granularity. Unfortunately, there is no way to do better.

**Theorem 2** For any tile \( T \) with \( B \) vertices on the dependency graph for one dimensional Haar wavelets, we must have \( u(T) \leq 1 + \log B \)

**Proof:** We proceed by induction on the tile size, \( B \). The base case of \( B = 1 \) is trivially \( u(T) = 1 = 1 + \log B \).

Now me make the inductive step, and assume that the result is true for all tiles \( T^* \) with \( |T^*| < B \). Consider any tile \( T \subseteq V \) with \( |T| = B \). Let \( r \in V \) be the root of the subtree generated by \( T \), and denote the part of \( T \) that lies in the left and right subtrees of \( r \) by \( T_L \) and \( T_R \) respectively. Notice that we must have \( |T_R|, |T_L| < B \), therefore the inductive hypothesis applies.

First we will assume that both \( |T_L| \) and \( |T_R| \) are nonzero, the exceptional case will be handled below. Denote \( \alpha = |T_R|/|T| < 1 \) and let \( \beta \) be the probability that a randomly selected source that lies below \( T \) is on the right side of \( r \). Then

\[
\begin{align*}
u(T) & \leq \beta u(T_R) + (1 - \beta) u(T_L) + 1 \\
& \leq \beta [1 + \log(\alpha|T|)] + (1 - \beta)[1 + \log((1 - \alpha)|T|)] \\
& = 1 + \log |T| + \beta \log \alpha + (1 - \beta) \log(1 - \alpha) \\
& < 1 + \log |T|
\end{align*}
\]

Now we catch the exception. If one of \( T_L \) and \( T_R \) is empty then we must have \( r \in T \) since \( r \) is the root of the smallest subtree containing \( T \). In this situation, clearly \( \beta = 1/2 \) and the size of the non-empty subtile (say it is \( T_R \)) is \( |T_R| = |T| - 1 \). Thus

\[
u(T) = 1 + \beta u(T_R) \leq 1 + \frac{1}{2} (\log(|T| - 1) + 1) < 1 + \log |T|
\]

when \( |T| \geq 2 \). So the exception has been handled, and the theorem proved. \( \Box \)

It seems that this limit on average utility of a tile must also place limits on the cost of a wavelet overlap query. This is the content of the next result.

**Corollary 2** For all tilings with tiles of size \( B \) on a domain of size \( N \), sampling sources uniformly, the average cost of a wavelet overlap query is at least \( \log_{2} B N \).
Figure 3: An example of nine points (marked with “x”) stored on a 2D disk block as product of two 1D virtual blocks (each marked with a triangle)

Proof:

\[ E_s[e(s)] = \frac{1}{N} \sum_{s \in D} e(s) = \frac{1}{N} \sum_{T \in T} S(T) \]
\[ \geq \frac{1}{N} \sum_{T \in T} \sum_{s \in D} \frac{u(T, s)}{\lg B + 1} \]
\[ = \sum_{s \in D} \frac{\lg N}{N(\lg B + 1)} \]
\[ = \lg_{2B} N \]

6 Multiple Dimensions

The optimal disk block allocation for one dimensional wavelets can be used to construct optimal allocations for (tensor product) multivariate wavelets. We simply decompose each dimension into optimal virtual blocks, and take the Cartesian products of these virtual blocks to be our actual blocks. To illustrate, consider the following example.

Example 3 Consider a two dimensional domain \((d = 2)\) where the size of each dimension is \(N = 8\). The Haar dependency tree for each dimension is depicted in Figure 3a. For each node, we also illustrate its offset in a one-dimensional array. Suppose the size of 2D disk blocks is 9. Thus, we need to choose two 1D virtual blocks of size 3 and generate the product. For example, as shown in Figure 3a, the wavelets at offset \(\{2, 4, 5\}\) make one optimal virtual block for one dimension, and the wavelets at \(\{3, 6, 7\}\) make another optimal virtual block for the other dimension. Figure 3b illustrates what the product of these two blocks looks like in a 2D array, marking with an “x” the points that belong to the product block.

This multi-dimensional block arrangement allows us to immediately obtain the following result.
Theorem 3 In a $d$-dimensional domain of size $N^d$, when wavelet coefficients are stored using the Cartesian product optimal block allocation, the I/O cost of the multi-dimensional wavelet overlap query in a domain of size $N$ is $O((\log_B N^d) N)$. Furthermore, if the dimensions all have the same size then for any Cartesian product block strategy, the average I/O cost will be $\Omega((\log_B N^d) N)$.

We conjecture that this lower bound holds for all block allocations, not just Cartesian product allocations. In any case, this allows us to answer polynomial range-sum queries quickly:

Corollary 3 When wavelet coefficients are stored using the Cartesian product optimal block allocation, the I/O cost of polynomial range-sum query evaluation is $O(2^d \log_B N^d)$. If the dimensions all have size $N$, then the I/O cost will also have the lower bound $\Omega(2^d (\log_B N - 1)^d)$.

7 Extending ProPolyne

The ProPolyne’s progressive query answering strategies presented in [37, 36] are based on the observation that some wavelet coefficients are much more important for a query answer than others. This allows us to compute the importance of each record to be retrieved and fetch the most important items first. The approximate answers returned progressively minimize both average and worst case error.

The results above show that we can achieve considerable savings by placing wavelet data on disk blocks in an appropriate way and retrieving coefficients at block level granularity. In order to obtain progressive query results in this setting, we need to compute the importance of disk blocks rather than individual wavelets.

When evaluating a single range-sum query, the importance of a data wavelet coefficient can be defined as the (square of the) magnitude of the corresponding query wavelet coefficient. In other words, denoting the importance of a wavelet $\xi$ by $i(\xi)$ and denoting the wavelet coefficients of a query vector $q$ by $\hat{q}(\xi)$, we can take $i(\xi) = |\hat{q}(\xi)|^2$.

An inspection of the proofs of Theorems 1 and 2 in [36] shows that there are two natural ways to extend this definition to disk blocks: we can either choose the importance function to minimize worst-case error, or we can choose it to minimize average error. In particular, we define two block-importance functions

$$i_2(B) = \sum_{\xi \in B} |\hat{q}(\xi)|^2 \quad i_\infty(B) = \max_{\xi \in B} |\hat{q}(\xi)|$$

and adapt the arguments in [36, 37] to prove the following result.

Theorem 4 (Informal) By fetching disk blocks in decreasing order of $i_\infty$-importance, we progressively minimize the worst case error of our query approximation. By fetching disk blocks in decreasing order of $i_2$-importance, we progressively minimize the average square error of our query approximation.
These results extend in a natural way to provide control of structural error in batch query answers [36], and we have implemented a batch query answering system that retrieves blocks in order to minimize the mean square error. We present experimental results using this system in Section 8, but we defer the detailed theorems to the full paper.

8 Experimental Results

We have implemented ProPolyne based on the ideas presented in this paper and are using it to develop web service access to climate grid datasets provided to us by our colleagues at NASA’s Jet Propulsion Laboratories. The system is implemented in C# using an ODBC connection to a SQL server to store and access wavelet data (see Figure 4). This section provides an overview of how block allocation choices affect the performance of this system. All experiments were performed on a sample dataset with 15.7 million records and four dimensions: altitude, latitude, longitude, and time. There are 16 altitudes, 128 latitudes, 64 longitudes, and 122 time points, recording temperature every 12 hours during March and April 2001. Temperatures are measured in degrees Kelvin. Our system is designed to efficiently answer batches of aggregate queries over ranges that partition the data domain, providing users with a summary view of the entire data set.

The results reported here are based on a randomly generated workload of 100 batch queries, where each batch query partitions the latitude dimension into 8 ranges, and partitions the altitude, longitude, and time dimensions into 4 ranges. The range-sum queries request the average temperature per each range. Dimensions were partitioned into ranges uniformly, so that all possible partitions are equally likely. We have experimented with different batch sizes and found similar results, with the following exceptions. For smaller

---

Note that these experiments are very different than those reported in [37, 36] because the underlying storage system maintains wavelets on disk blocks rather than individual coefficients.
batches all methods produced very accurate progressive results quickly, even though the I/O cost varied dramatically. This happens because the ranges in a small batch are large, and queries are well approximated by a small number of low resolution wavelets. For large batches, the optimal placement technique still provides much better progressive estimates, but there is very little difference in I/O performance— for a large batch you are essentially scanning the entire dataset, disk placement cannot be of great benefit.

8.1 Comparisons

The problem addressed in this paper is novel, and thus we lack appropriate direct comparisons. However there are two "common sense" block allocations which provide enlightening benchmarks. One of these allocations is obtained by essentially storing the data in row-major order on disk. The other captures the locality effects of a space-filling curve.

The first technique is the most naive: the multi-dimensional wavelet coefficients can be laid out naturally in a one dimensional array (in fact this is how we compute them in the first place [31]). We simply cut this array into contiguous subarrays of equal size to obtain our disk blocks. Specifically, if we denote the value of the $i$-th item in the $b$-th block by $v(b, i)$ and denote the array of wavelet coefficients by $w$ then we have

$$v(b, i) = w[B * b + i]$$

for blocks of size $B$. This is (approximately) the allocation we would obtain by default when we store the wavelet data as a one dimensional array on disk and let the file system determine the block allocation. We call this the naive allocation.

Our next technique has more respect for the multi-dimensional structure. Now we think of the wavelet coefficients as lying multi-dimensional array, and slice the data domain into cubes by slicing each dimension into virtual blocks. Each cell in this slicing will correspond to a disk block and every wavelet coefficient in the cell will be stored in the block. This gives us a natural multi-index for our disk blocks and a natural multi-index for wavelet coefficients in each block. Again, denoting the value at location $i = (i_0, i_1, \ldots, i_{d-1})$ in block $b = (b_0, b_1, \ldots, b_{d-1})$ by $v(b, i)$, we have

$$v(b, i) = w[(B_0 b_0 + i_0, B_1 b_1 + i_1, \ldots, B_{d-1} b_{d-1} + i_{d-1})]$$

where the slices are of size $B_i$ in dimension $i$. We call this the slice and dice allocation.

Another obvious tool to use for block allocations is a space-filling curve [28]. However for our application we have no need to place a one dimensional index structure on our coefficients, and the slice and dice allocation captures spatial locality at least as well as any space-filling curve. We use the slice and dice strategy as a proxy for space filling curves.

8.2 I/O Reduction

When answering these batch queries, the number of disk blocks fetched to answer a query using the wavelet-optimal allocation is substantially less than the number retrieved using
Figure 5: Mean number of blocks needed for exact query answer for a variety of block sizes and allocation techniques.

alternate methods. In Figure 5 we show the average number of blocks needed to evaluate the batch queries in our generated workload. Results are shown for the optimal, slice, and naive methods for a variety of block sizes. The block sizes were chosen so that we could produce optimal blocks consisting of complete subtree tiles.

The most important observation is that our optimal strategy outperforms the other two strategies, often by a factor of three or four. The theory suggests that the optimal strategy should always give the best results, but this evidence demonstrates that the difference is substantial. Another interesting and unexpected observation is that the naive block allocation consistently outperforms the slicing strategy. For all strategies, the number of blocks retrieved is only a small fraction of the total number of blocks in the database. This is because wavelets provide very efficient exact answers to ad hoc range-sum queries [37, 36].

Another interesting and unexpected observation is that the naive block allocation strategy outperforms the slicing strategy. This could be due to the fact that the naive blocks typically contain both high and low resolution data, while many of the domain slicing blocks contain only spatially localized high resolution data. When these high resolution blocks are retrieved, there are typically only a small number of useful items. Low resolution data are more likely to be useful.

8.3 Progressive Query Approximation

Our earlier work on ProPolyne has shown that wavelets provide excellent approximations of the matrices that arise in evaluation of batch range-sum queries [37, 36]. We noted above that for any allocation to disk blocks, we can compute the importance of a block so that by performing the most important I/O first we minimize either worst case or average error.
Although we have proven that our dependency graph based allocation is the best possible for exact query answering, other strategies may provide better approximate answers early in the computation. The results in this section suggest that this is not the case, and that the optimal placement gives us superior approximate results from the first block onward.

We measure the approximation quality in two ways. Our first approach is data independent. We use \( \nu_2 \), the importance function to minimize mean square error, and normalize so that the total importance of all columns in each query is one. We measure the cumulative importance of all retrieved disk blocks as the query progresses, aggregate the results over our randomly generated query workload, and report the results in Figure 6.

The optimal strategy dominates from the beginning, but it is interesting to note how poor a job the naive approximation does compared to the others, even though it is substantially better than the slicing method for exact query answering. As with the exact query analysis we performed these tests for a variety of block sizes, but saw no qualitative differences.

Our second measure of the approximation quality measures the accuracy of query answers on a real data set. We also progressively evaluated the queries on the temperature dataset described above, and report the mean relative error as a function of the number of disk blocks retrieved in Figure 6(b). The results are not surprising in light of Figure 6(a). The optimal placement performs extremely well, providing a mean relative error under 0.1% after retrieving only 10 disk blocks. The slice blocks also perform quite well, with mean relative error under 0.1% after retrieving 125 disk blocks. The naive approach performs terribly- mean relative error does not fall below 100% until the query evaluation is almost complete! Still we see that after retrieving about 500 disk blocks, the error for the naive
approach falls of rapidly. This is because the naive approach only requires an average of 387 blocks to answer the query exactly. The reduction comes almost entirely from the fact that as queries results become exact, the relative error becomes zero.

9 Conclusion

In this paper, we first briefly reviewed our prior work on ProPolyne, which is an efficient wavelet-based technique for exact, approximate and progressive evaluation of range-aggregate queries. Next, we identified a unique access pattern that arises when evaluating point and range queries on wavelet data with ProPolyne in particular but even more importantly with any other wavelet-based data management technique in general. We proved that these query types place a serious limit on our ability to effectively cluster wavelet data. Hence, we designed an allocation strategy with performance close to this theoretical limit, and showed that it leads to significant improvements in performance of an implemented query answering system. Finally, we extended ProPolyne exploiting our wavelet disk placement technique in order to design disk block-aware progressive query answering strategies that deliver excellent approximate results for hard queries after a small number of disk accesses.

We note that there are many other forms of multi-resolution and pre-aggregated data that may have access patterns similar to wavelets. We believe that the techniques proposed in this paper will also produce good disk placement techniques in these areas.

10 Future Work

Future work is plentiful. Below is a non-exhaustive list of our future plans in this area. This list proves that we have just scratched the surface and this very fruitful research area still has several other interesting and open problems for investigation.

Sparse wavelets: This paper deals with dense wavelet data, avoiding the problems solved by techniques such as space-filling curve based indices [28]. Real data will often be sparse, and large wavelet synopses of massive data sets may not fit well in main memory. Thus we need to use the observations of this paper to find ways to handle sparse wavelet data. Our first effort will be on developing dependency graph-filling curves.

I/O efficient wavelet transformation and updates: Our current implementation of ProPolyne operates on datasets that have been transformed off-line and prepared for the system. Moreover, no incremental update of the data set is supported. For our large datasets, the transformation sometimes takes days to complete. To make ProPolyne practical, its web-services should be extended with transformation and update web-services. This, however, is more than a simple implementation effort. The challenge is that current wavelet transformation techniques assume data can be fit into the main memory during the transformation. However, for large datasets, an I/O efficient transformation technique is required. We intend to build upon the lifting scheme [1] and extend it with an appropriate buffer management technique so that the intermediate wavelet arrays can be broken into chunks that can be
read and written optimally to minimize I/O during transformation. Similarly, as new data sets become available, the update web-service should be able to update the coefficients without requiring a full transformation. We currently have one approach where this can be achieved if the total size of the data set is known a priori. We need to extend this work for cases where the ultimate size is not known.

Error bound guarantees and optimal ordering of coefficients: In order to support approximate queries for scientific applications, it is desirable for the user to get informed of an error bound for any given approximate query result (or during the progression of the result). For example, the user may trade-off time for accuracy, i.e., prior to the query execution, given an error tolerance, the user can see the approximate response time for a query or vice-versa. Current utilization of wavelets for data compression assumes that always the entire signal needs to be reconstructed and hence suggests dropping low energy coefficients to obtain the best $L^2$ norm error. However, once wavelets are used to approximate range queries, depending on the range, we may observe a huge variation on the error. The error depends heavily on whether the corresponding coefficients for a given range query are kept or not. A very recent study [13] proposes a new approach in dropping wavelet coefficients that would result in guaranteed error bounds for range and point queries. We intend to build upon this work by modifying it for ProPolyne. The idea is to instead of dropping coefficients, use the technique to sort the corresponding data coefficients for a given ProPolyne query. This way, the approximate and progressive query features of ProPolyne will be realized while an error bound can be provided along the way. Recall that ProPolyne suggests an ordering for “query” coefficients. We plan to modify the technique proposed in [13] in order to suggest a priority or “weight” for sets of data coefficients. The combination of the ProPolyne query coefficients ordering and the weights of their corresponding data coefficients would provide us with a new ordering that will take into consideration the importance of both data and query. Hence, we believe that this new framework would not only provide us with error bounds but also result in better query approximations. The other advantage of this framework is that by ignoring ProPolyne query ordering we can mimic the traditional wavelet data compression techniques and by ignoring the data coefficient weights, we mimic the pure ProPolyne technique.

Hybridization of ProPolyne: We intend to generalize the mechanism underlying ProPolyne by looking beyond pure wavelets to find other basis which may be more effective on a particular dataset or for a particular query workload. Not only do query evaluation algorithms need to be developed in this setting, but there is also a need for best-basis (or at least good-basis) algorithms that efficiently select an appropriate basis from a library of possibilities. As a first step in this direction we want to develop a hybrid version of ProPolyne which uses the standard basis in a subset of the dimensions (the standard dimensions) and uses wavelets in all other dimensions. To illustrate, consider a database of ground-station data with schema $<\text{ground-station-id}, \text{latitude}, \text{longitude}, \text{time}, \text{height-variation}>$. Suppose the stations have fixed locations (i.e., lats and longs). Hence, if we project away the time and height-variation dimensions, we will have a relatively small result set. Consequently, we may want to use the standard basis (i.e., no transform) on the small relation $<\text{ground-station-id}, \text{latitude}, \text{longitude}>$ and use wavelets on the others. Given this decomposition of the dimensions, relational selection and aggregation operators can be used in the standard
dimensions to accumulate the results of ProPolyne queries in the other dimensions. Clearly
the best choice of hybridization will perform at least as well as a pure relational algorithm
or pure ProPolyne. Our preliminary analysis indicates that for many realistic datasets
and query patterns, hybridizations can perform dramatically better. The challenge here is
making the correct choice of standard dimensions. We would like to develop one algorithm
which efficiently identifies good dimension decompositions as part of the database population
process, and a complementary algorithm which selects the most appropriate available
basis to use for evaluation of a particular query. The basis library used by this hybrid
algorithm is a subset of the full wavelet packet basis library (DWPT). DWPT [42] is a
generalization of wavelet transform that includes wavelet coefficients as well as summary
and details of details at different levels. Hence, by recursively applying a summary and a
detail filter on both summaries and details, DWPT quickly computes a large amount of
information about the space and frequency characteristics of a function at different scales.
Not only will the techniques developed here be valuable in practice, our understanding of
this simplified problem will provide a foundation for future use of the full wavelet packet
transform (DWPT).

ProPolyne for general relational algebra operators: Finally, we intend to generalize the
applicability of the principles underlying ProPolyne. While range aggregate queries are
useful, linear algebraic approximation can be used for much more general types of queries.
Towards this end, we intend to extend our work on batch of range queries [36] which require
the simultaneous evaluation of multiple related range aggregates. These queries are very
common and include SQL group-by queries, drill-down queries, or general MDX expressions.
The key observation there is that these queries act as linear maps where range queries act
as linear functionals. Thus, where we approximate a vector to estimate a range query
result, we must approximate a matrix to estimate a general query result. We propose
novel techniques to select bases in which these matrices are very sparse, giving natural
query evaluation algorithms with low computational complexity. With these bases in hand,
we have developed query evaluation algorithms which share I/O maximally and retrieve
the most important data first in order to provide fast approximate results. This work on
batch queries [36] has helped us in understanding the mechanics of matrix approximation
for approximate query answering; at the same time it has provided insight into appropriate
error measures. Relational algebra operators also have matrix representations, and once we
have a thorough understanding of how matrix approximation works in the simpler setting
described above, we will be prepared to develop and analyze fundamentally novel exact,
progressive, and approximate evaluation strategies for relational algebra queries.

11 Acknowledgments

The authors would like to thank the following students for their help in development of
ProPolyne: Mehrdad Jahangiri, Dimitris Scharadis and Mehdi Sharifzadeh.
References


