IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITJ2783

Hybrid Query and Data Ordering for Fast and Progressive Range-Aggregate Query Answering

Cyrus Shahabi, University of Southern California, USA
Mehrdad Jahangiri, University of Southern California, USA
Dimitris Sacharidis, University of Southern California, USA

ABSTRACT

Data analysis systems require range-aggregate query answering of large multidimensional datasets. We provide the necessary framework to build a retrieval system capable of providing fast answers with progressively increasing accuracy in support of range-aggregate queries. In addition, with error forecasting, we provide estimations on the accuracy of the generated approximate results. Our framework utilizes the wavelet transformation of query and data hypercubes. While prior work focused on the ordering of either the query or the data coefficients, we propose a class of hybrid ordering techniques that exploits both query and data wavelets in answering queries progressively. This work effectively subsumes and extends most of the current work where wavelets are used as a tool for approximate or progressive query evaluation. The results of our experimental studies show that independent of the characteristics of the dataset, the data coefficient ordering, contrary to the common belief, is the inferior approach. Hybrid ordering, on the other hand, performs best for scientific datasets that are inter-correlated. For an entirely random dataset with no inter-correlation, query ordering is the superior approach.

Keywords: approximate query; multidimensional dataset; OLAP; progressive query; range-aggregate query; wavelet transformation

INTRODUCTION

Modern data analysis systems need to perform complex statistical queries on very large multidimensional datasets; thus, a number of multivariate statistical methods (e.g., calculation of covariance or kurtosis) must be supported. On top of that, the desired accuracy varies per application, user, and/or dataset, and it can well be traded off for faster response time. Furthermore, with progressive queries—a.k.a., *anytime* algorithms (Grass & Zilberstein, 1995; Bradley, Fayyad, & Reina, 1998) or *online* algorithms (Hellerstein, Haas, & Wang, 1997)—during the query run time, a measure of the quality of the running answer, such as error forecasting, is required.

We believe that our methodologies described in this article can contribute towards this end.

We propose a general framework that utilizes the wavelet decomposition of multidimensional data, and we explore progressiveness by selecting data values to retrieve based on some *ordering* function. The use of the wavelet decomposition is justified by the well-known fact that the query cost is reduced from query size to the logarithm of the data size, which is a major benefit especially for large-range queries. The main motivation for choosing ordering functions to produce different evaluation plans lies in the observation that the order in which data is retrieved from the database has an impact on the accuracy of the intermediate results (see Experimental Results). Our work, as described in this article, effectively extends and generalizes most of the related work where wavelets are used as a tool for approximate or progressive query evaluation (Vitter, Wang, & Iyer, 1998; Vitter & Wang, 1999; Lemire, 2002; Wu, Agrawal, & Abbadi, 2000; Schmidt & Shahabi, 2002; Garofalakis & Gibbons, 2002).

Our general query formulation can support any high-order polynomial range-aggregate query (e.g., variance, covariance, kurtosis, etc.) using the joint data frequency distribution similar to Schmidt and Shahabi (2002). However, we fix the query type at range-sum queries to simplify our discussion. Furthermore, the main distinction between this article and the work in Schmidt and Shahabi (2002) is that they only studied the ordering of *query* wavelets, while in this article we discuss a general framework that includes the ordering of not only *query* and *data* wavelets, but also the hybrid of the two.

Let us note that our work is not a simple application of wavelets to scientific datasets. 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 correlation 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 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 sort the coefficients and specify which coefficients to keep or drop. Even at query time, we need to retrieve all required data coefficients to plan the optimal order. Thus, we study alternative ways of ordering both the query and data coefficients to achieve optimal progressive (or approximate) answers to polynomial queries. The major observation from our experiments is that no matter whether the data is compressible or not, ordering data coefficients alone is the inferior approach.

CONTRIBUTIONS

The main contributions of this article are as follows:

- Introduction of Ordering Functions that measure the significance of wavelet coefficients and form the basis of our framework. Ordering functions formalize and generalize the methodologies commonly used for wavelet approximation; however, following our definition, they can be used to provide either exact, approximate, or progressive views of a dataset. In addition, depending on the function used, ordering functions can provide deterministic bounds on the accuracy of the dataset view.
- Incorporation of existing wavelet approximation techniques (first-B, highest-B) to our framework. To our knowledge, the only other work that takes under consideration the significance of a wavelet coefficient in answering arbitrary range queries is that of Garofalakis and Gibbons (2002). We have modified their Low-Bias Probabilistic Wavelet Synopses technique to produce the MEOW ordering function, which measures significance with respect to a predefined set of queries (workload).
- Definition of the query vector, the data vector, and the answer vector (hypercubes in the multidimensional case). By ordering any of these vectors, we construct different progressive evaluation plans for query answering. One can either apply an ordering function for the query or answer vector online as a new query arrives, or can apply an ordering function to the data vector offline as a preprocessing step. We prove that online ordering of the answer vector results in the ideal, yet not feasible, progressive evaluation plan.
- Proposition of Hybrid Ordering, which uses a highly compact representation of the dataset to produce an evaluation plan that is very close to the ideal one. Careful utilization of this compact representation leads to our Hybrid* Ordering Algorithm.
- Conducting several experiments with large real-world and synthetic datasets (of over 100 million values) to compare the effectiveness of our various proposed ordering techniques under different conditions.

The remainder of the article is organized as follows. In next section we present previous work on selected OLAP topics that are closely related to our work. Next, we introduce the ordering functions that our framework is built upon and the ability to forecast some error metrics. Later, we present our framework for providing progressive answers to Range-Sum queries, and we prescribe a number of different techniques. We examine all the different approaches of our framework on very large and real multidimensional datasets, and draw useful conclusions for the applicability of each approach in the experimental results section. Finally in the last section, we conclude and sketch our future work on this topic.

RELATED WORK

Gray, Bosworth, Layman, and Pirahesh (1996) demonstrated the fact that analysis of multidimensional data was inadequately supported by traditional relational databases. They proposed a new relational aggregation operator, the *data cube*, that accommodates aggregation of multidimensional data. The relational model, however, is inadequate to describe such data, and an inherent multidimensional approach using sparse arrays was suggested in Zhao, Deshpande, and Naughton (1997) to compute the data cube. Since the main use of a data cube is to support aggregate queries over ranges on the domains of the dimensions, a large amount of work has been focused on providing faster answers to such queries at the expense of higher update and maintenance cost. To this end, a number of pre-aggregation techniques were proposed. Ho, Agrawal, Megiddo, and Srikant (1997) proposed a data cube (Prefix Sum) in which each cell stored the summation of the values in all previous cells, so that it can answer a range-sum query in constant time (more precisely, in time $O(2^d)$). The update cost, however, can be as large as the size of the cube. Various techniques (Geffner, Agrawal, Abbadi, & Smith, 1999; Chan & Ionnidis, 1999) have been proposed that balance the cost of queries and updates.

For applications where quick approximate answers are needed, a number of different approaches have been taken. Histograms (Poosala & Ganti, 1999; Gilbert, Kotidis, Muthukrishnan, & Strauss, 2001; Gunopulos, Kollios, Tsotras, & Domeniconi, 2000) have been widely used to approximate the joint data distribution and therefore provide approximate answers in aggregate queries. Random sampling (Haas & Swami, 1995; Gibbons & Matias, 1998; Garofalakis & Gibbons, 2002) has also been used to calculate synopses of the data cube. Vitter and colleagues have used the wavelet transformation to compress the prefix sum data cube (Vitter et al., 1998) or the original data cube (Vitter & Wang, 1999), constructing compact data cubes. Such approximations share the disadvantage of being highly data dependent and as a result can lead to bad performance in some cases.

The notion of progressiveness in query answering with feedback, using running confidence intervals, was introduced in Hellerstein et al. (1997) and further examined in Lazaridis and Mehrotra (2001) and Riedewald, Agrawal, and Abbadi (2000). Wavelets and their inherent multi-resolution property have been exploited in providing answers that progressively get better. In Lemire (2002) the relative prefix sum cube is transformed to support progressive answering, whereas in Wu et al. (2000) the data cube is directly transformed. Let us note that the type of queries supported in a typical OLAP system is quite limited. One exception is the work of Schmidt and Shahabi (2002), where general polynomial range-sum queries are supported using the transformation of the joint data distribution into the wavelet domain.

Ordering Wavelet Coefficients

The purpose of this section is to define *ordering functions* for wavelet datasets which form the foundation for the proposed framework.

Ordering Functions and Significance

In this section, we order wavelet coefficients based on the notion of "significance." Such an ordering can be utilized for either approximating or getting progressively better views of a dataset.

A *Data Vector* of size N will be denoted as $\mathbf{d} = (d_1 d_2 \cdots d_N)$, whereas a *Wavelet Vector* of size N is $\mathbf{w} = (w_1 w_2 \cdots w_N)$. These two vectors are associated with each other through the *Discrete Wavelet Transform*: $DWT(\mathbf{d}) = \mathbf{w} \Leftrightarrow DWT^{-1}(\mathbf{w}) = \mathbf{d}$.

Given a wavelet vector $\mathbf{w} = (w_1 w_2 \cdots w_N)$ of size N, we can define a total order $\geq$ on the set of wavelet coefficients $W = (w_1, w_2, ..., w_n)$. The totally ordered set $\{w_{\sigma 1}, w_{\sigma 2}, ..., w_{\sigma N}\}$ has the property $\forall_{\sigma i} \leq \sigma_j, w_{\sigma i} \geq w_{\sigma j}$. The indexes σ_i define a permutation σ of the wavelet coefficients that can be expressed through the *Mapping Vector* $\mathbf{m} = (\sigma_1, \sigma_2, ..., \sigma_N)$.

A function $f : W \rightarrow R$ that obeys the property $w_i \geq w_j \Leftrightarrow f(w_i) \geq f(w_j)$ is called an *Ordering Function*. This definition suggests that the sets $(W, \geq)$ and $(f(W), \geq)$ are order isomorphic—that is, they result in the same permutation of wavelet coefficients. With the use of ordering function, the semantic of the $\geq$ operator and its effect is more clearly portrayed. Ordering functions assign a real value to a wavelet coefficient which is called *Significance*.

At this point we should note that for each total order $\geq$, there is a corresponding class of ordering functions. However, any representative of this class is sufficient to characterize the ordering of the wavelet coefficients.

Common Ordering Functions

In the database community wavelets have been commonly used as a tool for approximating a dataset (Vitter et al., 1998; Vitter & Wang, 1999). Wavelets indeed possess some nice properties that serve the purpose of compressing the stored data. This process of compression involves dropping some "less" significant coefficients. Such coefficients carry little information, and thus setting them to zero should have little effect on the reconstruction of the dataset. This is the intuition used for approximating a dataset.

The common question that a database administrator may face is: Given a storage capacity that allows storing only $B << |W|$ coefficients of the wavelet vector W, what is the best possible B-length subset of W? We shall answer this question in different ways, with respect to the semantics of the phrase *best possible subset*.

We can now restate two widely used methods using the notion of ordering functions.

First-B: The most significant coefficients are the lowest frequency ones. The intuition of this approach is that high-frequency coefficients can be seen as noise and thus they should be dropped. One simple ordering function for this method could be $f_{FB}(w_i) = -i$, which leads to a linear compression rule. The First-B ordering function has the useful property that it is dependant on the indices of the coefficients.

Highest-B: The most significant coefficients are those that have the most energy. The energy of a coefficient w_i is defined as $E(w_i) = |w_i|^2$. The intuition comes from signal processing: the highest powered coefficients are the most important ones for the reconstruction of the transformed signal which leads to the common non-linear "hard thresholding" compression rule. One ordering function therefore is $f_{HB}(w_i) = |w_i|^2$.

Extension to Multidimensional Hypercubes

Up to now, we were constricting ourselves to the one-dimensional case, where the data is stored in a vector. In the general case the data is of a multidimensional nature and can be viewed as a multidimensional hypercube (Gray et al., 1996). Each cell in such an n-dimensional (hyper-)cube contains a single data value and is indexed by n numbers, each number acting as an index in a single dimension. Without loss of generality, each of the n dimensions of the cube **d** have S distinct values and are indexed by the integers 0,1, ..., $S-1$ so that the size of the cube is $|\mathbf{d}| = S^n$. The decomposition of the data cube into the wavelet domain is done by the application of the one-dimensional DWT on the cube for one dimension, then applying DWT on the resulting cube for the next dimension and continuing for all dimensions. We will use $\hat{\mathbf{d}}$ to denote the transformed data cube:

$$\hat{\mathbf{d}} = DWT_n \cdot DWT_{n-1} \cdots DWT_1(\mathbf{d})$$

Applying an ordering to a multidimensional cube is no longer a permutation on the domain of the indices, but it is rather a multidimensional index on the cube. The result of ordering a hypercube is essentially a one-dimensional structure, a vector. Each element of the mapping vector **m** is a set of n indices, instead of a single index in the one dimensional case. To summarize, the ordered vector together with the mapping vector is just a different way to describe the elements in the cube, with the advantage that a measure of significance is taken under consideration. One last issue that we would like to discuss is the definition of the First-B ordering of the previous section. It is not exactly clear what the First-B ordering in higher dimensions should be, but we clarify this while staying in accordance with our definition. We would like the lower frequency coefficients to be the most significant ones. Due to the way we have applied DWT on the cube, the lower frequency coefficients are placed along the lower sides of the cube. So, for an n-dimensional cube of size S, at each dimension we define a First-B ordering function for the element at cell $(i_0, i_1, ..., i_{n-1})$ as:

$$f_{FB_1}(d_{i_0,i_1,\ldots,i_{n-1}}) = \\ -[S \cdot \min(i_0, i_1, \ldots, i_{n-1}) + (i_0 + i_1 + \ldots + i_{n-1})]$$

However, one could choose a slightly different ordering function that starts at the lowest-indexed corner and visits all nodes in a diagonal fashion until it reaches the highest-indexed one:

$$f_{FB_2}(d_{i_0,i_1,\ldots,i_{n-1}}) = -(i_0 + i_1 + \ldots + i_{n-1})$$

Minimum Error Ordering in the Wavelet Domain (MEOW)

Formerly, we redefined the two most widely used ordering functions for wavelet vectors. These methods however can produce approximations that, in some cases, are very inaccurate, as illustrated in Garofalakis and Gibbons (2002). The authors show that for some queries the approximate answer can vary significantly compared to the actual answer, and they propose a probabilistic approach to dynamically select the most "significant" coefficients. In this section, we construct an ordering function termed MEOW based on these observations such that it fits into our general framework. We assume that a query workload has been provided in the form of a set of queries, and we try to find the ordering that minimizes some error metric when queries similar to the workload are submitted. We refer to the workload, that is a set of queries, using the notation . This set contains an arbitrary number of point-queries and range-queries.

In real-world scenarios, such a workload can be captured by having the system run for a while, so that queries can be profiled, similar to a typical database system profiling for future optimization decision. A query on the data cube **d** will be denoted by q. The answer to this query is $D\{q\}$. Answering this query with a lossless wavelet transformation of the data cube yields the result $W_N\{q\} = D\{q\}$, where $N = |\mathbf{d}|$ is the size of the cube. Answering the query with a B-approximation (B coefficients retained) of the wavelet transformation yields the result $W_B\{q\}$. We use the notation *err*(*org, rec*) to refer to the error introduced by approximating the original (*org*) value by the reconstructed approximate (*rec*) value. For example the relative error metric would be:

$$err_{rel}(org, rec) = \frac{|org - rec|}{org}$$

MEOW assigns significance to a coefficient with respect to how bad a set of queries would be answered if this coefficient was dropped; in other words, the most significant coefficient is the one in which the highest aggregated error occurs across all queries. Towards this end, a list of candidate coefficients is initialized with all the wavelet coefficients. At each iteration the most significant coefficient is selected to be maintained and hence removed from the list of candidates, so that the next-best coefficient is always selected in a greedy manner. Note that removing the most significant coefficient from the candidate list is equivalent to maintaining that coefficient as part of the B-approximation. More formally:

$$f_{MEOW}(w_i) =$$

$$\underset{\forall q \in QS}{agg}\,(err(D\{q\}, (W_{N-M} \setminus \{w_i\})\{q\}))$$

where $N = |\mathbf{d}|$ is the size of the cube; W_{N-M} is the (N-M)-coefficient wavelet approximation of the data vector (running candidate list), which is the result of selecting the M most significant coefficients seen so far; $W_{N-M} \setminus \{w_i\}$ is the (N-M-1)-coefficient wavelet approximation, which is the result of additionally selecting the coefficient under consideration. The function is used for aggregating the error values occurred for the set of queries; any L^p norm can be used if the error values are considered as a vector.

Ordering Functions and Error Forecasting

We have seen that obtaining an ordering on the wavelet coefficients can be useful when we have to decide the best-B approximation of a dataset. Having an ordering also helps us answer a query in a progressive manner, from the most significant to the least significant coefficient. The reason behind this observation is the fact that the ordering function assigns a value to each coefficient, namely its significance. If the ordering function is related to some error metric that we wish to minimize, the significance can be further used for error forecasting. That is, we can foretell the value of an error metric by providing a strict upper bound to it. In this section, we shall see why such a claim is plausible.

HIGHEST-B: The ordering obtained for Highest-B minimizes the L^2 error between the original vector and its B-length approximation. Let **w** be a vector in the wavelet domain and $\mathbf{w_B}$ its highest B approximation. Then the L^2 error is given by the summation of the power of the coefficients not included in the B approximation.

$$|\mathbf{w} - \mathbf{w_B}|^2 = \sum_{i=B+1}^{|\mathbf{w}|} w_i^2$$

This implies that including the highest B coefficients in the approximation minimizes the L^2 error for the reconstruction of the original wavelet vector. In progressive query evaluation we can forecast an error metric as follows. Suppose we have just retrieved the k-th coefficient w whose significance is $|w_k|^2$. The L^2 error is strictly less than the significance of this coefficient multiplied by the number of the remaining coefficients:

$$|\mathbf{w}-\mathbf{w}_B|^2 \le (|W|-k) \cdot f_{HB}(w_k)$$

Notice that in this case the forecasted error is calculated on the fly. We simply need to keep track of the number of coefficients we have retrieved so far.

MEOW: The ordering function used in MEOW is directly associated with an error metric that provides guarantees on the error occurred in answering a set of queries. Thus we can immediately report the significance of the last coefficient retrieved as the error value that we forecast. There are several issues that arise from this method, the most important being the space issue of storing the significance of coefficients. For the time being, we shall ignore these issues and assume that for any coefficient we can also look up its significance. Consequently, we can say that if the submitted query q is included in the query set $q \in QS$ and the L^∞ norm is used for aggregation, then the error for answering this query is not more than the significance of the current coefficient w_k retrieved.

$$err(D\{q\}, W_k\{q\}) \le f_{MEOW}(w_k)$$

ANSWERING RANGE-SUM QUERIES WITH WAVELETS

In this section, we provide a definition for the range-sum query, as well as a general framework for answering queries in a progressive manner where results get better as more data is retrieved from the database.

Range-Sum Queries and Evaluation Plans

Let us assume that the data we will deal with is stored in an n-dimensional hypercube **d**. We are interested in answering queries that are defined on an n-dimensional hyper-rectangle R. We define the general type of queries that we would like to be capable of answering.

Definition 1: A range-sum query $Q(R,\mathbf{d})$ of range R on the cube **d** is the summation of the values of the cube that are contained in the range. Such a query can be expressed by a cube of the same size as the data cube that has the value of one in all cells within the range R and zero in all cells outside that range. We will call this cube the query cube **q**.

The answer to such a query is given by the summation of the values of the data cube **d** for each cell ξ contained in the range:

$$a = \sum_{\xi \in R} \mathbf{d}(\xi).$$

We can rewrite this summation as the multiplication of a cell in the query cube **q** with a corresponding cell in the data cube **d**.

$$a = \sum_{\xi \in \mathbf{d}} \mathbf{q}(\xi) \cdot \mathbf{d}(\xi) \qquad (1)$$

As shown in Schmidt and Shahabi (2002), Equation 1 is a powerful formalization that can realize not only SUM, COUNT, and AVG, but also any polynomial up to a specific degree on a combination of attributes (e.g., VARIANCE and COVARIANCE).

Let us provide a very useful lemma that applies not only for the Discrete Wavelet Transform, but also for any transformation that preserves the Euclidean norm.

Lemma 1: If $\hat{\mathbf{a}}$ is the DWT of a vector **a** and $\hat{\mathbf{b}}$ is the DWT of a vector **b** then

$$< \mathbf{a}, \mathbf{b} > = < \hat{\mathbf{a}}, \hat{\mathbf{b}} > \Leftrightarrow \sum_i \mathbf{a}[i] \cdot \mathbf{b}[i] = \sum_j \hat{\mathbf{a}}[j] \cdot \hat{\mathbf{b}}[j]$$

We borrowed the following theorem from Schmidt and Shahabi (2002) to justify the use of wavelets in answering range-sum queries.

Theorem 1: The n-dimensional query cube **q** of size S in each dimension defined for a range R can be transformed into the wavelet domain in time $O(2^n log^n S)$, and the number of non-zero coefficients in $\hat{\mathbf{q}}$ is less than $O(2^n log^n S)$. This theorem results in the following useful corollary.

Corollary 1: The answer to a range-sum query $Q(R,\mathbf{d})$ is given by the following summation:

$$a = \sum_{\xi \in \hat{\mathbf{d}}} \hat{\mathbf{q}}(\xi) \cdot \hat{\mathbf{d}}(\xi)$$

and can be computed in $O(2^n log^n S)$ retrieves from the database.

This corollary suggests that an exact answer can be given in $O(2^n log^n S)$ steps. We can exploit this observation to answer a query in a progressive manner, so that each step produces a more precise evaluation of the actual answer.

Definition 2: A *progressive evaluation plan* for a range-sum query

$$Q(R,\mathbf{d}) = \sum_{\xi \in \hat{\mathbf{d}}} \hat{\mathbf{q}}(\xi) \cdot \hat{\mathbf{d}}(\xi)$$

is an n-dimensional index σ of the space defined by the cube $\hat{\mathbf{d}}$. The sum at the j-th progressive step

$$\tilde{a}_\sigma(j) \equiv \tilde{Q}(R,\mathbf{d}) = \sum_{\xi_\sigma = 0}^{j} \hat{\mathbf{q}}(\xi_\sigma) \hat{\mathbf{d}}(\xi_\sigma)$$

is called the *approximate answer* at iteration *j*.

It should be clear that if the multidimensional index visits only the non-zero query coefficients then approximate answer converges to the actual answer as *j* reaches $O(2^n log^n S)$. We would like to find an evaluation plan that causes the approximate answer to converge fast to the actual value. Towards this end, we will use an error metric to measure the quality of the approximate answer.

Definition 3: Given a range-sum query, an *optimal progressive evaluation plan* σ_0 is a plan that at each iteration produces an approximate answer $\tilde{a}$ that is closest to the actual answer *a*, based on some error metric, when compared to any other evaluation plan σ_i at the same iteration.

$$err(a, \tilde{a}_{\sigma_o}(j)) \le err(a, \tilde{a}_{\sigma_i}(j)), \forall j, \forall i \neq o$$

One important observation is that the n-dimensional index σ of a progressive evaluation plan defines three single-dimensional vectors—the query vector, the data vector, and another vector that we call the answer vector. The answer vector is formed by the multiplication of a query with a corresponding data coefficient.

Definition 4: The multiplication of a query coefficient with a data coefficient yields an answer coefficient. These answer coefficients form the *answer vector* **a**:

$$\mathbf{a}[i] = \hat{\mathbf{q}}[i] \cdot \hat{\mathbf{d}}[i]$$

Note that the answer vector is *not* a wavelet vector. In order to answer a query, we must sum across the coefficients of the answer vector.

Definition 5: Given the permutation σ of a progressive evaluation plan, we can redefine the approximate answer $\tilde{a}$ as the summation of a number of answer coefficients.

$$\tilde{a}_{\sigma}(j) = \sum_{i=0}^{j} \mathbf{a}[i]$$

Similarly, the actual answer is given by the following equation.

$$a = \sum_{i=0}^{|\mathbf{a}|} \mathbf{a}[i]$$

To summarize up to this point, we have defined the data cube, the query cube, and their wavelet transformations, and have seen that by choosing a multidimensional index, these cubes can be viewed as single-dimensional vectors. Finally we have defined the answer vector for a given range-sum query. In the next section we will explore different progressive evaluation plans to answer a query. By choosing an ordering for one of the three vectors described above—the query vector, the data vector, and the answer vector—we can come up with different evaluation plans for answering a query. Furthermore, we have a choice of different ordering functions for the selected vector.

Ordering of the Answer Vector

In this section, we will prove that a progressive evaluation plan that is produced by an ordering of the answer vector is the optimal one, yet not practically applicable. Recall that we have defined optimality in terms of obtaining an approximate answer as close to the actual answer as possible. Thus, the objective of this section can be restated as following: given an answer vector what is the permutation of its indices that yields the closest-to-actual approximate answer.

The careful reader should immediately realize that we have already answered this question. If we use the L^2 error norm to measure the quality of an approximate answer, then the ordering of choice has to be Highest-B. Please note that although we have defined Highest-B for a vector whose coefficients correspond to the wavelet basis, the property of this ordering applies for any vector defined over orthonormal basis; the standard basis is certainly orthonormal.

The L^2 error norm of an approximate answer is:

$$|a - \tilde{a}|^2 = \sum_{i=j+1}^{|\mathbf{a}|} (\mathbf{a}[i])^2$$

Therefore we have proven that the Highest-B ordering of the answer vector results in an optimal progressive evaluation plan. However, there is one serious problem with this plan. We have to first have the answer vector to order it, which means we have to retrieve all data coefficients from the database. This defeats our main purpose of finding a good evaluation plan so that we can progressively retrieve data coefficients. Therefore, we can never achieve the optimal progressive evaluation plan, but we can try to get close to it, and henceforth we use the Highest-B ordering of the answer vector as our lower bound.

Since the answer vector will not be available at query time, we can try to apply an ordering on either the query or the data cube. This is the topic of the following sections.

Ordering of the Query Cube

We will answer a range-sum query with an evaluation plan that is produced by ordering the query cube. Since we are not interested in the data cube, the evaluation plan will be generated at query time and will depend only on the submitted query. Below are the steps required to answer any range-sum query.

Preprocessing Steps (Off-line)

1. Transform the data cube $\mathbf{d}$ in the wavelet domain and store it in that form $\hat{\mathbf{d}}$.

Answering a Range-Sum Query (Online)

1. Construct the query cube and transform it into the wavelet domain.

$$\hat{\mathbf{q}} \leftarrow DWT(\mathbf{q})$$

2. Apply an ordering function to the transformed query cube $\hat{\mathbf{q}}$ to obtain the mapping vector (m-vector).

$$\mathbf{m} \leftarrow order(\hat{\mathbf{q}})$$

3. Iterate over the m-vector, retrieve the data coefficient corresponding to the index stored in the m-vector, multiply it with the query coefficient, and add the result to the approximate answer.

$$a = \sum_{i \in \mathbf{m}} \hat{\mathbf{q}}[i] \cdot \hat{\mathbf{d}}[i]$$

It is important to note that the number of non-zero coefficients is $O(2^n log^n S)$ for an n-dimensional cube of size S in each dimension which dramatically decreases the number of retrieves from the database and thus the cost of answering a range-sum query. This is the main reason why we use wavelets.

The ordering functions that we can use are First-B (Wu et al., 2000) and Highest-B. We cannot conclude which of the two ordering functions would produce a progressive evaluation plan that is closer to the optimal one for an arbitrary data vector. The Highest-B ordering technique suggested in Schmidt and Shahabi (2002) yields overall-better results only when we average across all possible data vectors, or equivalently when the dataset is completely random. In Experimental Results we will see the results of using both methods on different datasets.

Ordering of the Data Cube

In this section, we will answer a range-sum query using an evaluation plan that is produced by applying an ordering function on the data cube. We will provide the steps required and discuss some difficulties that this technique has. First, let us present the algorithm to answer a query in a similar way to query ordering.

Preprocessing Steps (Off-line)

1. Transform the data cube $\mathbf{d}$ in the wavelet domain and store it in that form $\hat{\mathbf{d}}$.
2. Apply an ordering function on the transformed data cube to obtain the mapping vector.

$$\mathbf{m} \leftarrow order(\hat{\mathbf{d}})$$

Answering a Range-Sum Query (Online)

1. Construct the query cube and transform it into the wavelet domain.

$$\hat{\mathbf{q}} \leftarrow DWT(\mathbf{q})$$

2. Retrieve from database a value from the m-vector and retrieve the data coefficient corresponding to that value. Multiply it with the query coefficient and add the result to the approximate answer. Repeat until all values in the m-vector have been retrieved.

$$a = \sum_{i \in \mathbf{m}} \hat{\mathbf{q}}[i] \cdot \hat{\mathbf{d}}[i]$$

There is a serious difference between this algorithm and the query ordering algorithm; the m-vector is stored in database and cannot be kept in memory, since it is as big as the data cube. This means that we can only get one value of the m-vector at a time together with the corresponding data coefficient. Besides the fact that we are actually retrieving two values at each iteration, the most important drawback is the fact that the approximate answer converges to the actual answer much slower, as it needs all data coefficients to be retrieved, thus time of $O(S^n)$.

This problem has been addressed in Vitter et al. (1998) and Vitter and Wang (1999) with the Compact Data Cube approach. The main idea is to compress the data by keeping only a sufficient number of coefficients based on the ordering method used. The resulting cube is so small that it can fit in main memory and queries can be answered quickly. This, however, has the major drawback that the database can support only approximate answers.

Last but not least, if the significance of a coefficient is stored in the database, then it can be used to provide an error estimation as discussed earlier. The data cube coefficients are partitioned in disjoint sets (levels) of decreasing significance, with one significance value assigned to each level (the smallest significance across all coefficients of the same level). When the last coefficient of a level is retrieved, the significance of that level is also retrieved and used to estimate the associated error metric. When our framework is used for progressive answering, the error metric provides an estimation of how good the answer is. Furthermore, the level error metrics can be used to calculate an estimation of the time our methodology needs to reach a certain level of accepted inaccuracy for a submitted query, before actually retrieving data from the database.

Hybrid Ordering

We have seen that the best progressive evaluation plan results from ordering the answer cube with the Highest-B ordering function. This, however, is not feasible since it requires that all the relevant data coefficients are retrieved, which comes in direct contrast with the notion of progressiveness. Also, we have seen how we can answer a query by applying an ordering to either the query or the data cube. The performance of these methods depends largely on the dataset and its compressibility. In this section, we will try to take this dependence out of the loop and provide a method that performs well in all cases.

With this observation in mind, it should be clear that we wish to emulate the way the best evaluation plan is created, which is ordering the answer vector. To create the answer vector, we need the query cube, which is available at the query time, and the data cube, which is stored in the database. As noted earlier, we cannot have both of these cubes in memory to do the coefficient-by-coefficient multiplication and then order the resulting vector. However we can use a very compact representation of the data cube, construct an approximate answer vector, and come up with an ordering that can be used to retrieve coefficients from the database in an optimal order. This sentence accurately summarizes the intuition behind our hybrid ordering method.

Hybrid Ordering Algorithms

For our hybrid method we need a very compact, yet close representation of the data cube transformed into the wavelet domain. We will call this cube the *approximate transformed data cube* $\tilde{\mathbf{d}}$. This cube must be small enough so that it can be fetched from the database in only a few retrieves. We will come back shortly to discuss the construction of such a cube; for the time being this is all we need to know for describing the algorithm of our hybrid method.

Preprocessing Steps (Off-line)

1. Transform the data cube $\mathbf{d}$ in the wavelet domain and store it in that form $\hat{\mathbf{d}}$.
2. Create a compact representation $\tilde{\mathbf{d}}$ of the data cube; $\tilde{\mathbf{d}} \cong \hat{\mathbf{d}}$.

Answering a Range-Sum Query (Online)

1. Construct the query cube and transform it into the wavelet domain:

$$\hat{\mathbf{q}} \leftarrow DWT(\mathbf{q})$$

2. Retrieve the necessary data from the database to construct the approximate transformed data cube $\tilde{\mathbf{d}}$.
3. Create the approximate answer cube by multiplying element by element the query cube and the approximate data cube.

$$\tilde{\mathbf{a}}[i] = \hat{\mathbf{q}}[i] \cdot \tilde{\mathbf{d}}[i]$$

4. Apply the Highest-B ordering function to the approximate answer cube $\tilde{a}$ to obtain the mapping vector $\mathbf{m}$.
5. Iterate over the m-vector, retrieve the data coefficient corresponding to the index stored in the m-vector, multiply it with the query coefficient, and add the result to the approximate answer.

$$a = \sum_{i \in \mathbf{m}} \hat{\mathbf{q}}[i] \cdot \hat{\mathbf{d}}[i]$$

There is an online overhead associated with the additional retrieval cost for constructing the approximate transformed data cube, since data has to be retrieved from the database. This cost however can be amortized for a batch of queries. In addition this approximate data cube can be used to provide an approximation of the actual answer. The Hybrid* Ordering algorithm takes advantage of this observation and consequently converges faster to the actual answer.

The initial value of *ans* is no longer zero, but is equal to an approximation of the answer constructed from the approximate data cube $\sum \tilde{\mathbf{d}}[i] \cdot \hat{\mathbf{q}}[i]$. Later, each time we retrieve an actual data coefficient $\hat{\mathbf{d}}[i]$ from the database, the value in *ans* is corrected by adding the error of the approximation $\hat{\mathbf{d}}[i] \cdot \hat{\mathbf{q}}[i] - \tilde{\mathbf{d}}[i] \cdot \hat{\mathbf{q}}[i]$.

Assuming that the approximate data cube is memory resident (which is the case for batched queries for example), the Hybrid* Algorithm can give an approximate answer without any retrieves from the database, while correcting the initial approximation with each retrieval. In fact, Hybrid and Hybrid* Orderings, as we demonstrate in the Experimental Results section, outperform other orderings for scientific datasets since they approximate optimal ordering by exploiting both query and data information.

Approximating the Transformed Data Cube

In this section, we will construct a different approximate transformed data cube to be used by our Hybrid* Algorithm; we defer the performance comparison to the next section. The purpose of such a cube is not to provide approximate answers, but rather to accurately capture the trend of the transformed data cube, so that we can use its approximation to produce an evaluation plan close to optimal. The method of constructing an approximate data cube must also be able to easily adapt to changes in the desired accuracy.

Equi-Sized Partitioning of the Untransformed Data cube (EPU): The *untransformed* data cube is partitioned into k^n equi-sized partitions and the average of each partition is selected as a representative. Each partition in EPU consists of the same value, which is the average, so that the total different values stored are only k^n. The transformation of this cube into the Wavelet Domain is also compact since there are at most $k^n log^n S$ non-zero coefficients in the approximate transformed data cube resulting from this partitioning scheme.

Equi-Sized Partitioning of the Transformed Data cube (EPT): With EPT the *transformed* data cube is partitioned into k^n equi-sized partitions, and the average of each partition is selected as a representative. Again, each partition in EPT consists of the same value, which is the average, so that the total different values stored are only k^n.

Resolution Representatives (RR): This method differs from EPT in that the representatives are selected per resolution level, rather than per partition. For a one-dimensional vector, there are *logS* resolution levels, and in the *n*-dimensional case, there are $log^n S$ resolution levels. Thus, picking one representative per resolution level yields $log^n S$ distinct values. In the case where a less compact approximate data cube is desired, some representatives can be dropped; in the case where more accuracy is required, more representatives per resolution level can be kept.

Wavelet Approximation of the Transformed Data cube: The Wavelet Decomposition is used to approximate the data cube. Given a desired storage of *B* coefficients, either Highest-B (C-HB), any of the First-B ordering functions (f_{FB_1}, f_{FB_2}) (C-FB), or MEOW can be used to compress the data cube. The result consists of *B* distinct values.

EXPERIMENTAL RESULTS

First, we describe our experimental setup. Subsequently, we compare different evaluation plans and draw our main conclusions. Later, we compare and select the best approximation of the data cube. Finally we investigate how larger approximate data cubes effect the performance of our Hybrid* algorithm.

Experimental Setup

We report results from experiments on three datasets. TEMPERATURE and PRECIPITATION (Widmann & Bretherton, 1949) are real-world datasets, and RANDOM is a synthetic dataset. RANDOM, PRECIPITATION, and TEMPERATURE are incompressible, semi-compressible, and fully compressible datasets, respectively. Here, we use the term compressible to describe a dataset that has a compact, yet accurate wavelet approximation.

TEMPERATURE is a real-world dataset that measures the temperatures at points all over the globe at different altitudes for 18 months, sampled twice every day. We construct a four-dimensional cube with latitude, longitude, altitude, and time as dimension attributes, and temperature as the measure attribute. The corresponding size of the domain of these dimensions are 64,128,16, and 1024 respectively. This results in a dense data cube of more than 134 million cells.

RANDOM is a synthetic dataset of the same size as TEMPERATURE where each cell was assigned with a random number between 0 and 100. The result is a completely random dataset, which cannot be approximated.

PRECIPITATION is a real-life dataset that measures the daily precipitation for the Pacific North West for 45 years. We build a three-dimensional cube with latitude, longitude, and time as dimensional attributes, and precipitation as the measure attribute. The corresponding size of these dimensions are 8, 8, and 16,384 respectively. This makes a data cube of one million cells.

The data cubes are decomposed into the wavelet domain using the multidimensional wavelet transformation. We generated random range-sum queries with a uniform distribution. We measure the Mean Relative Error across all queries to compare different techniques in the following sections.

Comparison of Evaluation Plans

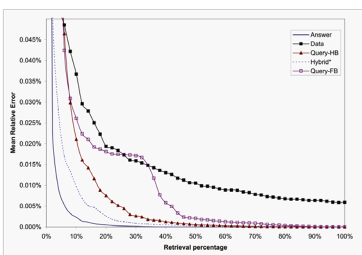
Figure 1 compares the performance of five evaluation plans. Please note that all graphs display the progressive accuracy of various techniques for queries on TEMPERATURE, RANDOM, and PRECIPITATION datasets. The horizontal axis always displays the number of retrieved values, while the vertical axis of each graph displays the mean relative error for the set of generated queries. Let us note that the online overhead of Hybrid* Ordering is not shown in the figures, as it applies only for the first query submitted. Because of its small size, it can be maintained in memory for all subsequent queries. The main observations are as follows.

Answer Ordering presents the lower bound for all evaluation plans, as it is an optimal, yet impractical evaluation plan. Answer ordering is used with the Highest-B ordering function, which leads to an optimal plan as discussed previously. (Hybrid* Ordering can outperform Answer Ordering at the beginning, because of the fact that it has precomputed an approximate answer.)

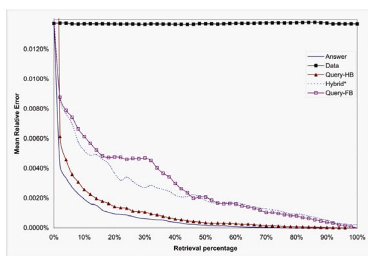
Data Ordering with the Highest-B ordering function performs slightly better than Query Ordering at the beginning, only for a compressible dataset like TEMPERATURE, but it can never answer a query with 100% accuracy using the same number of retrieves as the other methods. (Recall that the total number of required retrieves to answer any query is $O(S^n)$ for Data Ordering and is $O(2^n log^n S)$ for all other techniques). The use of the Highest-B ordering function is justified by the well-known fact that without query knowledge, Highest-B yields the lowest L^p error norm. When dealing with the RANDOM dataset or PRECIPITATION dataset, Data Ordering can never provide a good evaluation plan. Although we believe MEOW data ordering introduces improvement for progressive answering, we have not included MEOW ordering into the experiments with data ordering due to its large preprocessing cost. However, in the next section we use MEOW as a data cube approximation technique.

Query Ordering with Highest-B ordering function performs well with all datasets. Such an evaluation plan is the ideal

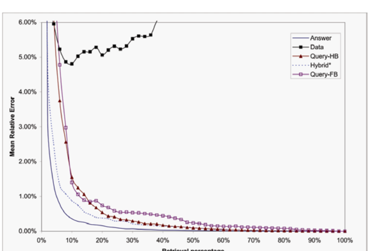
Figure 1. Comparison of evaluation plans



a. TEMPERATURE dataset

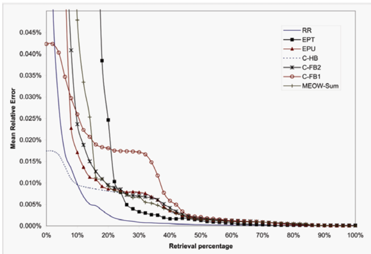


b. RANDOM dataset

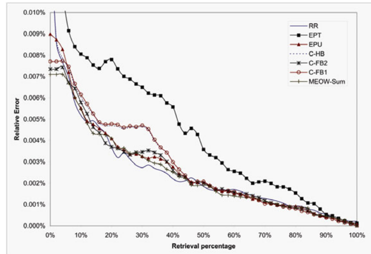


c. PRECIPITATION dataset

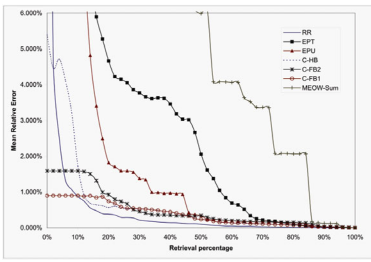
Figure 2. Hybrid ordering*



a. TEMPERATURE dataset



b. RANDOM dataset



c. PRECIPITATION dataset

when dealing with datasets that are badly compressible, but is inferior to Hybrid* for compressible datasets. We also see that Query Ordering with First-B, although trivial, can still perform well, especially with the PRECIPITATION dataset.

Hybrid* Ordering is the closest to optimal evaluation plan with a compressible dataset. In addition, this method has the advantage that it can immediately provide an answer with a reasonable error for a compressible dataset. However, Hybrid* does not introduce significant improvements over Query Ordering for an incompressible dataset. At this point, let us note that the approximate data cube was created using the Resolution Representatives (RR) method; the reason behind this is explained in the following section. The size of the approximate data cube is as small as 0.002% of the data cube size to emphasize both the ability to maintain such a structure in memory and the negligible retrieval cost. Later we will allow larger approximate data cubes. Let us indicate that Hybrid* Ordering subsumes and surpasses Hybrid Ordering; therefore, we do not report Hybrid Ordering in our experiments.

To conclude, *Hybrid* Ordering* is recommended for any real-life dataset that is compressible. *Query ordering*, on the other hand, can be the smartest choice for a completely incompressible dataset.

Choosing an Approximate Data Cube

In this section, we measure the performance of our Hybrid* Algorithm using our proposed techniques for creating approximate data cubes. Our purpose is to find the superior approximation of the dataset to use with Hybrid*.

Figures 2a and 2c suggest that Resolution Representatives performs better overall. However, when retrieving only a very small percentage of the data, the Wavelet Approximation techniques outperform Resolution Representatives. In particular, Highest-B ordering outperforms First-B ordering. As a rule of thumb, if the user-level application mostly needs extremely fast but approximate results, Wavelet Approximation should be used for creating an Approximate Data Cube. On the other hand, if the user-level application can tolerate longer answering periods, Resolution Representatives can provide more accurate answers.

We used MEOW with different L^p norm as the aggregation function among which L^1 norm (sum of errors) performs the best. As illustrated in Figure 2a, MEOW with the L^1 norm performs badly at the beginning, but soon catches up with the other Wavelet Approximation techniques. Figure 2c clearly shows the poor performance of EPT and MEOW for a semi-compressible dataset.

When dealing with a complete random dataset, Figure 2b implies that the previous observations still hold. Resolution Representatives still produce overall better results. Among the Wavelet Approximation techniques, the First-B ordering methods consistently outperform Highest-B. The reason behind this lies in the fact that RANDOM is not compressible (all wavelet coefficients are equally "significant"). The previous observations also hold for the PRECIPITATION dataset (see Figure 2c).

The most important conclusion one can draw from these figures is that Resolution Representatives is the best performing technique for constructing approximate data cubes. Hybrid* ordering evaluation plans that utilize Resolution Representatives produce good overall results, even in the worst case of a very badly compressible dataset.

Performance of Larger Approximate Data Cubes

In this section, we investigate the performance improvement one can expect by using Hybrid* Ordering with a larger approximate data cube. Recall that in our experiments we used an approximate data cube that is 50,000 times smaller than the actual cube (percentage of the cube). We created the approximate data cubes using the Resolution Representatives method, as previously recommended.

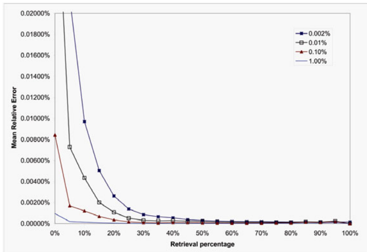
Figure 3a illustrates that for a compressible dataset, the more coefficients we keep, the better performance the system will have. In contrast, Figure 3b shows that keeping more coefficients for an incompressible dataset does not result in any improvement. This is because such a highly random dataset cannot have a good approximation, even when a lot of space is utilized. Figure 3c also shows a slight improvement with larger approximate cubes.

In sum, we suggest that the size of the approximate data cube is a factor that should be selected with respect to the type of the dataset. For compressible datasets, Hybrid* Ordering is more efficient with larger approximations.

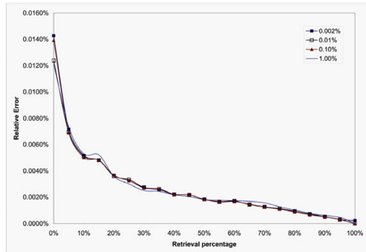
CONCLUSION AND FUTURE WORK

We introduced a general framework for providing fast and progressive answers for range-aggregate queries by utilizing the

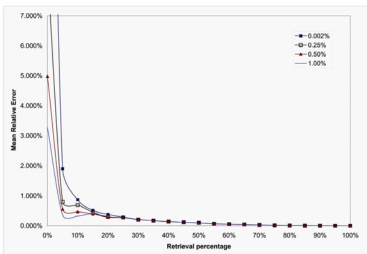
Figure 3. Performance of larger approximate data cubes



a. TEMPERATURE dataset



b. RANDOM dataset



c. PRECIPITATION dataset

wavelet transformation. We conducted extensive comparisons among different ordering schemes and prescribed solutions based on factors such as the compressibility of the dataset, the desired accuracy, and the response time. In sum, our proposed Hybrid* algorithm results in the minimal retrieval cost for real-world datasets with (even slight) inter-correlation. In addition, its utilization of a small approximate data cube as the initial step generates a good and quick approximate answer with minimum overhead. Another main contribution of this article is the observation that data-only ordering of wavelets results in the worst retrieval performance, unlike the case of the traditional usage of wavelets in signal compression applications. Our current and future work is oriented towards investigating other factors, such as the type and frequency of queries, and the amount of available additional storage.

REFERENCES

Bradley, P.S., Fayyad, U.M., & Reina, C. (1998). Scaling clustering algorithms to large databases. *Knowledge Discovery and Data Mining*, 9-15.

Chan, C.-Y. & Ionnidis, Y.E. (1999). Hierarchical cubes for range-sum queries. *Proceedings of the 25th International Conference on Very Large Data Bases* (pp. 675-686).

Garofalakis, M. & Gibbons, P.B. (2002). Wavelet synopses with error guarantees. *Proceedings of SIGMOD 2002.* ACM Press.

Geffner, S., Agrawal, D., Abbadi, A.E., & Smith, T. (1999). Relative prefix sums: An efficient approach for querying dynamic OLAP data cubes. *Proceedings of the 15th International Conference on Data Engineering* (pp. 328-335). IEEE Computer Society.

Gibbons, P.B. & Matias, Y. (1998, June 2-4). New sampling-based summary statistics for improving approximate query answers. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 331-342), Seattle, Washington, USA. ACM Press.

Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., & Strauss, M.J. (2001). Optimal and approximate computation of summary statistics for range aggregates. *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (pp. 228-237).

Grass, J. & Zilberstein, S. (1995). *Anytime algorithm development tools*. Technical Report No. UM- CS-1995-094.

Gray, J., Bosworth, A., Layman, A., & Pirahesh, H. (1996). Datacube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. *Proceedings of the 12th International Conference on Data Engineering* (pp. 152-159).

Gunopulos, D., Kollios, G., Tsotras, V.J., & Domeniconi, C. (2000). Approximating multi-dimensional aggregate range queries over real attributes. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 463-474).

Haas, P.J. & Swami, A.N. (1995, March 6-10). Sampling-based selectivity estimation for joins using augmented frequent value statistics. *Proceedings of the 11th International Conference on Data Engineering* (pp. 522-531), Taipei, Taiwan. IEEE Computer Society.

Hellerstein, J.M., Haas, P.J., & Wang, H. (1997). Online aggregation. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 171-182). ACM Press.

Ho, C., Agrawal, R., Megiddo, N., & Srikant, R. (1997). Range queries in OLAP data cubes. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 73-88). ACM Press.

Lazaridis, I. & Mehrotra, S. (2001). Progressive approximate aggregate queries with a multi-resolution tree structure. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 401-412).

Lemire, D. (2002, October). Wavelet-based relative prefix sum methods for range sum queries in data cubes. *Proceedings of CASCON 2002*.

Poosala, V. & Ganti, V. (1999). Fast approximate answers to aggregate queries on a data cube. *Proceedings of the 11th International Conference on Scientific and Statistical Database Management* (pp. 24-33). IEEE Computer Society.

Riedewald, M., Agrawal, D., & Abbadi, A.E. (2000). pCube: Update-efficient online aggregation with progressive feedback. *Proceedings of the 12th International Conference on Scientific and Statistical Database Management* (pp. 95-108).

Schmidt, R. & Shahabi, C. (2002). Propolyne: A fast wavelet-based technique for progressive evaluation of polynomial range-sum queries. *Proceedings of the Conference on Extending Database Technology* (EDBT'02). Berlin: Springer-Verlag (LNCS).

Vitter, J.S. & Wang, M. (1999). Approximate computation of multidimensional aggregates of sparse data using wavelets. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 193-204). ACM Press.

Vitter, J.S., Wang, M., & Iyer, B.R. (1998). Data cube approximation and histograms via wavelets. *Proceedings of the 7th International Conference on Information and Knowledge Management* (pp. 96-104). ACM.

Widmann, M. & Bretherton, C. (1949-1994). *50 km resolution daily precipitation for the Pacific Northwest.*

Wu, Y.-L., Agrawal, D., & Abbadi, A.E. (2000). Using wavelet decomposition to support progressive and approximate range-sum queries over data cubes. *Proceedings of the 9th International Conference on Information and Knowledge Management* (pp. 414-421). ACM.

Zhao, Y., Deshpande, P.M., & Naughton, J.F. (1997). An array-based algorithm for simultaneous multidimensional aggregates. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (pp. 159-170).

Cyrus Shahabi is currently an associate professor and director of the Information Laboratory with the Computer Science Department and is also a research area director a the NSF's Integrated Media Systems Center at the University of Southern California. He earned his PhD in computer science from the University of Southern California (August 1996). He has two books and more than 100 articles, book chapters, and conference papers in the areas of databases and multimedia. Dr. Shahabi is the recipient of the 2002 National Science Foundation CAREER Award and 2003 Presidential Early Career Awards for Scientists and Engineers (PECASE).

Dimitris Sacharidis was born in Athens, Greece. He earned a BS in electrical and computer engineering from the National Technical University of Athens (2001). In 2004, he earned an MS in computer science from the University of Southern California. Currently, he is a PhD student in the National Technical University of Athens, working on data streams analysis.

Mehrdad Jahangiri was born in Tehran, Iran. He earned a BS in civil engineering from the University of Tehran at Tehran (1999). He is currently working toward a PhD in computer science at the University of Southern California. He is also a research assistant working on multidimensional data analysis at Integrated Media Systems Center (IMSC) - Information Laboratory (InfoLAB) in the Computer Science Department of the University of Southern California.